THE APPLICATION OF KRIGING IN THE STATISTICAL ANALYSIS OF ANTHROPOMETRIC DATA
VOLUME III

THESIS

Michael Grant
Captain, USAF

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY
AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

(1)

AFIT/GOR/ENY/ENS/90M-8

THE APPLICATION OF KRIGING
IN THE STATISTICAL ANALYSIS
OF ANTHROPOMETRIC DATA
VOLUME III

THESIS

Michael Grant
Captain, USAF

AFIT/GOR/ENY/ENS/90M-8

Approved for public release; distribution unlimited

SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1a. REPORT SECURITY CLASSIFICATION UNCLASSIFIED	1b. RESTRICTIVE MARKINGS
2a. SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION/AVAILABILITY OF REPORT Approved for public release; distribution unlimited
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE	
4. PERFORMING ORGANIZATION REPORT NUMBER(S) AFIT/GOR/ENY/ENS/90M-8	5. MONITORING ORGANIZATION REPORT NUMBER(S)
6a. NAME OF PERFORMING ORGANIZATION School of Engineering 6b. OFFICE SYMBOL (If applicable) AFIT/ENY	7a. NAME OF MONITORING ORGANIZATION
6c. ADDRESS (City, State, and ZIP Code)	7b. ADDRESS (City, State, and ZIP Code)
8a. NAME OF FUNDING/SPONSORING ORGANIZATION AAMRL 8b. OFFICE SYMBOL (If applicable) HEG	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
8c. ADDRESS (City, State, and ZIP Code) AAMRL/HEG WPAFB, OH 45433	10. SOURCE OF FUNDING NUMBERS PROGRAM ELEMENT NO. / PROJECT NO. / TASK NO. / WORK UNIT ACCESSION NO.

11. TITLE (Include Security Classification)
THE APPLICATION OF KRIGING IN THE STATISTICAL ANALYSIS OF ANTHROPOMETRIC DATA

12. PERSONAL AUTHOR(S)
Michael Grant, B.S., M.S., Capt, USAF

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (Year, Month, Day)	15. PAGE COUNT
MS Thesis	FROM ______ TO ______		430

16. SUPPLEMENTARY NOTATION

17. COSATI CODES			18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB-GROUP	Kriging, Bayesian Statistics, Morphometrics, Geostatistics, Multivariate Analysis
12	03		

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

Thesis Advisors: David G. Robinson
Assistant Professor
Department of Aeronautics and Astronautics

Kenneth W. Bauer
Assistant Professor
Department of Operational Sciences

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT ☒ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT. ☐ DTIC USERS	21. ABSTRACT SECURITY CLASSIFICATION UNCLASSIFIED
22a. NAME OF RESPONSIBLE INDIVIDUAL David G. Robinson, Asst. Professor	22b. TELEPHONE (Include Area Code) (513) 255-2362 22c. OFFICE SYMBOL AFIT/ENY

DD Form 1473, JUN 86 — Previous editions are obsolete. — SECURITY CLASSIFICATION OF THIS PAGE

UNCLASSIFIED

Quality flight equipment is essential to flight crew safety and performance. Oxygen masks, night-vision goggles, and other apparatus must fit crew members comfortably and with complete functional precision. A problem currently facing the Air Force is the inconsistent quality of flight equipment. As new equipment is developed to improve crew members' performance, the requirement for design engineers to accurately account for the shape and variability of facial features becomes more critical.

This thesis develops the application of kriging in the statistical analysis of anthropometric data to support improvements in the design of flight equipment. Specifically, the geostatistical estimation technique of kriging is used to estimate the facial surfaces which influence the designs of flight apparatus. These surfaces account for the shape of the facial features and minimize the variance between individuals. A Kalman filter is developed to update and aggregate the kriged surfaces. As a proof of concept study, the techniques are demonstrated using data to support the design of the night-vision goggles currently under development. To further enhance the surface estimates, a multivariate analysis is performed to identify the factors which account for the majority of the variability between faces and to group the faces into homogenous clusters.

AFIT/GOR/ENY/ENS/90M-8

THE APPLICATION OF KRIGING IN THE STATISTICAL ANALYSIS OF ANTHROPOMETRIC DATA

VOLUME III

THESIS

Presented to the Faculty of the School of Engineering

of the Air Force Institute of Technology

Air University

In Partial Fulfillment of the

Requirements for the Degree of

Master of Science in Operations Research

Michael Grant, B.S., M.S.

Captain, USAF

March 1990

Approved for public release; distribution unlimited

Appendix F. *Data Alignment and Manipulation Programs*

The program contained in this appendix was used for labelling the coordinates of the aligned data points. The alignment program is not included in this thesis. However, the program and user documentation are available through Major David G. Robinson (AFIT/ENY).

Coordinate Labelling Program

This program reads from sXXX.align files and writes to the fXXX files.

Basically, the sXXX.align file contains the output from the alignment program. Each record consists of the angle, altitude, and radius (followed by three dummy variables which are not used) for a particular point. The following records are a sample of the s09.align file.

```
0.39 204.95 82.06 76.01 204.95 30.94
0.39 206.41 82.32 76.07 206.41 31.47
```

The fXXX file is used as input to the programs which structure the data in appropriately sized grids. The format for each record is as follows: row index, column index, value of x, value of y, the mean, variance, and the number of points in the block. The following records were extracted from f09.

```
19 34 0.7400 226.0000 103.39  0.21 11
19 35 0.7400 230.0000 103.48  0.11 9
```

To run the program for subject 09, type "fill.x list.09". The list.09 file consists of the following two records:

```
1
s09.align    12172
```

The 1 indicates that one file is read and the 12172 is the number of points in the file.

The code for this program begins on the following page .

```
#include <stdio.h>
#include <math.h>

/*   npts: number of total data points to be input
       for a single data set
    NX,NY: number of increments on the x,y axes
XMAX,YMAX: maximum values for x,y
XMIN,YMIN: minimum values for x,y
*/

#define NX 100
#define NY 50
#define XMAX 4.
#define XMIN 0.
#define YMAX 300.
#define YMIN 100.
#define DX (XMAX-XMIN)/NX
#define DY (YMAX-YMIN)/NY

struct point {
    float angle, latitude, radius;
    struct point *next;
    };

struct point *point_array[NX][NY];

main(argc,argv)
    int argc;
    char *argv[];
{
    int i,j;
    void Data_In(), Show_List();
    void Stat();

    Data_In(argv[1]);

    for(i=0; i<NX; i++) {
       for(j=0; j<NY; j++) {
/*        if(point_array[i][j] != NULL) {
             Show_List(i,j, point_array[i][j]);  */
             Stat(i,j, point_array[i][j]);
          /*}  */
```

```
            }
        }
    }
void Data_In(arg1)
/* reads data in format: angle, altitude, radius */
char *arg1;
{
    int i,j,m,n;
    int ii;
    float x,y,z;
    float dum1, dum2, dum3;
    int npts, nosub;  /* number of subject data files to be read */
    char subject_name[10];  /* name of subject data file to be read */

    FILE *fins, *fin;

    struct point *temp;

/* initializing pointers to NULL  */
    for (i=0; i<NX; i++)
        for(j=0; j<NY; j++)
            point_array[i][j] = NULL;

    fins = fopen(arg1,"r");
    fscanf(fins,"%d\n",&nosub);
    printf("A total of %d subject data files are to be read\n",nosub);


    for(ii=0;ii<nosub;ii++) {
       fscanf(fins,"%s %d\n",subject_name, &npts);
       printf("FIRST NOTICE: \n Reading %d data points\n from subject data
       file: %s \n",npts, subject_name);
       fin = fopen(subject_name,"r");

       for (i=0; i<npts; i++) {
         fscanf(fin,"%f %f %f %f %f %f\n", &x, &y, &z, &dum1, &dum2,
        &dum3);
/*         printf("%f %f %f %f %f %f\n", x,y,z,dum1,dum2,dum3);*/

        m = (int)(NY*(y-YMIN)/(YMAX-YMIN));
        n = (int)(NX*(x-XMIN)/(XMAX-XMIN));
/*         printf("location of point on grid: (m,n): %d %d \n",m,n);*/
```

```
      temp = (struct point *)malloc(sizeof(struct point));

      if((m>=0)&&(n>=0)){
         if (temp != NULL)  {
             temp->angle    = x;
             temp->latitude = y;
             temp->radius   = z;
             temp->next     = point_array[n][m];
             point_array[n][m] = temp;
         }
      }
      }
   fclose(fin);
   }
}
void Show_List(i,j,ps)
int i,j;
struct point *ps;
{
   float x,y;

   x= XMIN+(i-.5)*DX;
   y= YMIN+(j-.5)*DY;

   do
   {
     printf("%.4f %.4f %.2f %.2f %.2f\n", x, y, ps->angle, ps->latitude,
     ps->radius);

     ps = ps->next;
   } while (ps != NULL);
}

void Stat(i,j,ps)
int i,j;
struct point *ps;
{
   float x,y;
   float mean, var;
   float sum, sum2;
   int kntr;
```

```
    x= XMIN+(i-.5)*DX;
    y= YMIN+(j-.5)*DY;

    sum  = 0.;
    sum2 = 0.;
    kntr = 0.;
    mean = 0.;
    var  = 0.;

    if(ps != NULL ) {
      do
      {
          sum += (ps->radius);
          sum2 += (ps->radius * ps->radius);
          kntr++;
          ps = ps->next;
      } while (ps != NULL);

      mean = sum/kntr;
      var = sum2/kntr - (mean*mean);
      printf("%d %d %5.4f %5.4f %5.2f %5.2f %d\n", i,j, x, y, mean, var, kntr);
    }
    else
      printf("%d %d %.4f %.4f %.2f %.2f %d\n", i,j, x, y, mean, var, kntr);

}
```

Appendix G. *Structural Analysis Programs*

The programs contained in this appendix were used in performing the structural analysis of the data. Specifically, this appendix includes the programs for forming the grids, removing the trend, calculating the experimental variograms, and estimating the model parameters.

Model Fitting Program

The model fitting program is written in FORTRAN. Data is read from the output files of the experimental variogram program contained in the next section. Each record consists of h (distance), $\gamma(h)$, and the number of points used in determining $\gamma(h)$. The number of records for each subject is 40–4 directions times 10 lags. This program will also fit the models for the overall variogram. A program for consolidating the variogram data for any given number of subjects is provided later in this appendix. The output of this program includes the parameter estimates for the linear, De Wijsian, and spherical models and the associated r-squared values.

The code for this program begins on the next page.

```
      PROGRAM MAIN
C****************************************************************C
C*  THIS PROGRAM FITS A LINEAR, DE WIJSIAN, AND SPHERICAL MODEL *C
C*  TO THE VARIOGRAM DATA.  13 JAN 90.                          *C
C****************************************************************C
      COMMON/DAT1/H(1200), GAMMA(1200), WEIGHTS(1200), NPTS, IWGT
      COMMON/DAT2/Y(1200), W(1200), Z(1200,3), ZPW(3,1200)
      COMMON/LU/N, A(3,3), INDX(3), VV(3), B(3), D
      COMMON/OUT/BETA(3,3), RSQR(3), YHAT(3,1200), VAR, SPH1, SPH2
      CALL INITIAL
      CALL INPUT
      CALL SETUP
      CALL LINEAR
      CALL DEWJSN
      CALL SPHRCL
      CALL STATS
      CALL OUTPUT
      STOP
      END

      SUBROUTINE SETUP
C****************************************************************C
C*    THIS SUBROUTINE SETS UP THE COMMON MATRICES               *C
C****************************************************************C
      COMMON/DAT1/H(1200), GAMMA(1200), WEIGHTS(1200), NPTS, IWGT
      COMMON/DAT2/Y(1200), W(1200), Z(1200,3), ZPW(3,1200)
      SUM=0.0
      DO 10 I=1,NPTS
         Z(I,1)=1.0
         Y(I)=GAMMA(I)
         SUM=SUM+WEIGHTS(I)
   10 CONTINUE
      DO 30 I=1,NPTS
         DO 20 J=1,NPTS
            IF (I.EQ.J) THEN
               IF (IWGT.EQ.1) THEN
                  W(I)=WEIGHTS(I)/SUM
               ELSE
                  W(I)=1.0
               ENDIF
            ENDIF
   20    CONTINUE
```

```
   30 CONTINUE
      RETURN
      END

      SUBROUTINE LINEAR
C******************************************************************C
C*    THIS SUBROUTINE FITS A LINEAR MODEL.                         *C
C******************************************************************C
      COMMON/DAT1/H(1200), GAMMA(1200), WEIGHTS(1200), NPTS, IWGT
      COMMON/DAT2/Y(1200), W(1200), Z(1200,3), ZPW(3,1200)
      COMMON/LU/N, A(3,3), INDX(3), VV(3), B(3), D
      COMMON/OUT/BETA(3,3), RSQR(3), YHAT(3,1200), VAR, SPH1, SPH2
      DO 10 I=1,NPTS
         Z(I,2)=H(I)
   10 CONTINUE
      N=2
      CALL MATRIX
      BETA(1,1)=B(1)
      BETA(1,2)=B(2)
      RETURN
      END

      SUBROUTINE DEWJSN
C******************************************************************C
C*    THIS SUBROUTNE FITS A DE WIJSIAN MODEL TO THE DATA           *C
C******************************************************************C
      COMMON/DAT1/H(1200), GAMMA(1200), WEIGHTS(1200), NPTS, IWGT
      COMMON/DAT2/Y(1200), W(1200), Z(1200,3), ZPW(3,1200)
      COMMON/LU/N, A(3,3), INDX(3), VV(3), B(3), D
      COMMON/OUT/BETA(3,3), RSQR(3), YHAT(3,1200), VAR, SPH1, SPH2
      DO 10 I=1,NPTS
         Z(I,2)=LOG(H(I))
   10 CONTINUE
      N=2
      CALL MATRIX
      BETA(2,1)=B(1)
      BETA(2,2)=B(2)
      RETURN
      END

      SUBROUTINE SPHRCL
C******************************************************************C
```

```
C*    THIS SUBROUTINE FITS A SPHERICAL MODEL TO THE DATA        *C
C****************************************************************C
      COMMON/DAT1/H(1200), GAMMA(1200), WEIGHTS(1200), NPTS, IWGT
      COMMON/DAT2/Y(1200), W(1200), Z(1200,3), ZPW(3,1200)
      COMMON/LU/N, A(3,3), INDX(3), VV(3), B(3), D
      COMMON/OUT/BETA(3,3), RSQR(3), YHAT(3,1200), VAR, SPH1, SPH2
      DO 10 I=1,NPTS
         Z(I,2)=H(I)
         Z(I,3)=H(I)**3
   10 CONTINUE
      N=3
      CALL MATRIX
      DO 20 I=1,3
         BETA(3,I)=B(I)
   20 CONTINUE
      RETURN
      END

      SUBROUTINE MATRIX
C****************************************************************C
C*    THIS SUBROUTINE PERFORMS THE MATRIX MANIPULATIONS         *C
C****************************************************************C
      COMMON/DAT1/H(1200), GAMMA(1200), WEIGHTS(1200), NPTS, IWGT
      COMMON/DAT2/Y(1200), W(1200), Z(1200,3), ZPW(3,1200)
      COMMON/LU/N, A(3,3), INDX(3), VV(3), B(3), D
      DO 20 I=1,NPTS
         DO 10 J=1,N
            ZPW(J,I)=Z(I,J)*W(I)
   10    CONTINUE
   20 CONTINUE
      DO 60 J=1,N
         SUM1=0
         SUM2=0
         SUM3=0
         SUMY=0
         DO 50 I=1,NPTS
            SUM1=SUM1+ZPW(J,I)*Z(I,1)
            SUM2=SUM2+ZPW(J,I)*Z(I,2)
            SUM3=SUM3+ZPW(J,I)*Z(I,3)
            SUMY=SUMY+ZPW(J,I)*Y(I)
   50    CONTINUE
         A(J,1)=SUM1
```

```
         A(J,2)=SUM2
         A(J,3)=SUM3
         B(J)=SUMY
   60 CONTINUE
      CALL LUDCMP
      CALL LUBKSB
      RETURN
      END

      SUBROUTINE INITIAL
C****************************************************************C
C*    THIS ROUTINE ALLOWS THE USER TO ENTER THE FILE NAMES OF *C
C*    THE DATA FILES FROM THE TERMINAL OR TO READ THEM FROM   *C
C*    THE DEFAULT FILE (SPHERE.SET)                           *C
C****************************************************************C
      CHARACTER ANSWER*3, FILIN*32, FILOUT*32
      WRITE(6,*)'DO YOU WANT TO SPECIFY THE FILES FROM THE TERMINAL?'
      WRITE(6,*)'(ENTER YES IF SO; ELSE, DEFAULT = SPHERE.SET)'
      READ(5,'(A3)') ANSWER
      IF (ANSWER.EQ.'YES') THEN
          WRITE(6,*)'ENTER THE NAME OF THE INPUT FILE'
          READ(5,'(A32)') FILIN
          WRITE(6,*)'ENTER THE NAME OF THE OUTPUT FILE'
          READ(6,'(A32)') FILOUT
      ELSE
         OPEN(9,FILE='SPHERE.SET',STATUS='OLD')
         READ(9,'(A32)') FILIN
         READ(9,'(A32)') FILOUT
      ENDIF
      OPEN(10,FILE=FILIN,STATUS='OLD')
      OPEN(11,FILE=FILOUT,STATUS='NEW')
      RETURN
      END

      SUBROUTINE INPUT
C****************************************************************C
C*    THIS ROUTINE READS IN THE DATA FROM THE FILE SPECIFIED  *C
C*            H  - THE DISTANCE                               *C
C*      GAMMA(H) - THE VARIOGRAM                              *C
C*    WEIGHTS(H) -  THE NUMBER OF POINTS USED FOR GAMMA(H)    *C
C****************************************************************C
      COMMON/DAT1/H(1200), GAMMA(1200), WEIGHTS(1200), NPTS, IWGT
```

```
      IWGT=1
C      NPTS=1200
C      NPTS=300
C      NPTS=40
C      NPTS=1040
      NPTS=1000
      DO 10 I=1,NPTS
         READ(10,*) H(I), GAMMA(I), WEIGHTS(I)
   10 CONTINUE
      RETURN
      END

      SUBROUTINE LUDCMP
C******************************************************************C
C*  THIS SUBROUTINE WAS ADAPTED FROM NUMERICAL RECIPES, THE ART *C
C*  OF SCIENTIFIC COMPUTING, PP. 35-36.  THE ARRAY, A, IS RE-   *C
C*  ARRANGED AS ITS LU DECOMPOSITION.  INDX IS A VECTOR WHICH   *C
C*  RECORDS THE ROW PERMUTATION.  D IS + OR - TO INDICATE EVEN  *C
C*  OR ODD NUMBER OF ROW CHANGES.  THIS SUBROUTINE IS USED WITH *C
C*  LUBKSB.                                                     *C
C******************************************************************C
      PARAMETER (TINY=1.0E-20)
      COMMON/LU/N, A(3,3), INDX(3), VV(3), B(3), D
      D=1.
      DO 20 I=1,N
         AAMAX=0.
         DO 10 J=1,N
            IF (ABS(A(I,J)).GT.AAMAX) AAMAX=ABS(A(I,J))
   10    CONTINUE
         IF (AAMAX.EQ.0.) WRITE(11,*) 'SINGULAR MATRIX.'
         VV(I)=1./AAMAX
   20 CONTINUE
      DO 90 J=1,N
         IF (J.GT.1) THEN
            DO 40 I=1,J-1
               SUM=A(I,J)
               IF (I.GT.1) THEN
                  DO 30 K=1,I-1
                     SUM=SUM-A(I,K)*A(K,J)
   30             CONTINUE
                  A(I,J)=SUM
               ENDIF
```

```
40          CONTINUE
        ENDIF
        AAMAX=0.
        DO 60 I=J,N
          SUM=A(I,J)
          IF (J.GT.1) THEN
            DO 50 K=1,J-1
              SUM=SUM-A(I,K)*A(K,J)
50          CONTINUE
            A(I,J)=SUM
          ENDIF
          DUM=VV(I)*ABS(SUM)
          IF (DUM.GE.AAMAX) THEN
            IMAX=I
            AAMAX=DUM
          ENDIF
60      CONTINUE
        IF (J.NE.IMAX) THEN
          DO 70 K=1,N
            DUM=A(IMAX,K)
            A(IMAX,K)=A(J,K)
            A(J,K)=DUM
70        CONTINUE
          D=-D
          VV(IMAX)=VV(J)
        ENDIF
        INDX(J)=IMAX
        IF (J.NE.N) THEN
          IF (A(J,J).EQ.0.) A(J,J)=TINY
          DUM=1./A(J,J)
          DO 80 I=J+1,N
            A(I,J)=A(I,J)*DUM
80        CONTINUE
        ENDIF
90    CONTINUE
      IF (A(N,N).EQ.0.) A(N,N)=TINY
      RETURN
      END

      SUBROUTINE LUBKSB
C*****************************************************************C
C*  THIS SUBROUTINE WAS ADAPTED FROM NUMERICAL RECIPES THE ART *C
```

```
C*  OF SCIENTIFIC COMPUTING, PP. 36-37.  A AND INDX ARE DETER- *C
C*  MINED IN LUDCMP.  B IS THE RIGHT HAND SIDE VECTOR INITIAL- *C
C*  LY AND THEN BECOMES THE SOLUTION VECTOR.  THIS ROUTINE CAN *C
C*  BE CALLED SUCCESSIVELY WITH DIFFERENT B VECTORS.           *C
C*****************************************************************C
      COMMON/LU/N, A(3,3), INDX(3), VV(3), B(3), D
      II=0
      DO 20 I=1,N
         LL=INDX(I)
         SUM=B(LL)
         B(LL)=B(I)
         IF (II.NE.0)THEN
            DO 10 J=II,I-1
               SUM=SUM-A(I,J)*B(J)
   10       CONTINUE
         ELSE IF (SUM.NE.0.) THEN
            II=I
         ENDIF
         B(I)=SUM
   20 CONTINUE
      DO 40 I=N,1,-1
         SUM=B(I)
         IF (I.LT.N) THEN
            DO 30 J=I+1,N
               SUM=SUM-A(I,J)*B(J)
   30       CONTINUE
         ENDIF
         B(I)=SUM/A(I,I)
   40 CONTINUE
      RETURN
      END

      SUBROUTINE STATS
C*****************************************************************C
C*    THIS SUBROUTINE CALCULATES STATISTICS                        *C
C*****************************************************************C
      COMMON/DAT1/H(1200), GAMMA(1200), WEIGHTS(1200), NPTS, IWGT
      COMMON/DAT2/Y(1200), W(1200), Z(1200,3), ZPW(3,1200)
      COMMON/LU/N, A(3,3), INDX(3), VV(3), B(3), D
      COMMON/OUT/BETA(3,3), RSQR(3), YHAT(3,1200), VAR, SPH1, SPH2
      DOUBLE PRECISION SUM(3)
      DOUBLE PRECISION SUMT
```

```
      SUM1=0
      SUM2=0
      DO 10 I=1,NPTS
         SUM1=SUM1+Y(I)**2
         SUM2=SUM2+Y(I)
   10 CONTINUE
      RMEAN=SUM2/FLOAT(NPTS)
      VAR=(SUM1-(SUM2**2)/FLOAT(NPTS))/(FLOAT(NPTS)-1.0)
      SPH1=VAR-BETA(3,1)
      SPH2=(1.5*SPH1)/BETA(3,2)
      SUMT=0.0
      DO 20 I=1,NPTS
        SUMT=SUMT+(Y(I)-RMEAN)**2
        YHAT(1,I)=BETA(1,1)+BETA(1,2)*H(I)
        YHAT(2,I)=BETA(3,1)+BETA(3,2)*H(I)+BETA(3,3)*H(I)**3
C        YHAT(2,I)=BETA(2,1)+BETA(2,2)*LOG(H(I))
        IF(H(I).LT.SPH2) THEN
          YHAT(3,I)=SPH1*(1.5*H(I)/SPH2-.5*H(I)**3/SPH2**3)+VAR-SPH1
        ELSE
          YHAT(3,I)=VAR
        ENDIF
   20 CONTINUE
      DO 30 I=1,3
         SUM(I)=0
   30 CONTINUE
      DO 40 I=1,NPTS
         DO 40 J=1,3
            SUM(J)=SUM(J)+(Y(I)-YHAT(J,I))**2
   40    CONTINUE
   50 CONTINUE
      DO 60 I=1,3
         RSQR(I)=1.0-(SUM(I)/SUMT)
   60 CONTINUE
      RETURN
      END

      SUBROUTINE OUTPUT
C*****************************************************************C
C*    THIS ROUTINE OUTPUTS THE ESTIMATES                         *C
C*****************************************************************C
      COMMON/DAT1/H(1200), GAMMA(1200), WEIGHTS(1200), NPTS, IWGT
      COMMON/DAT2/Y(1200), W(1200), Z(1200,3), ZPW(3,1200)
```

```
      COMMON/LU/N, A(3,3), INDX(3), VV(3), B(3), D
      COMMON/OUT/BETA(3,3), RSQR(3), YHAT(3,1200), VAR, SPH1, SPH2
      IF(IWGT.NE.1) THEN
         WRITE(11,900)
         WRITE(11,910) BETA(1,1), BETA(1,2), RSQR(1)
         WRITE(11,920) BETA(2,1), BETA(2,2), RSQR(2)
         WRITE(11,930) SPH1, SPH2
         WRITE(11,940) SPH2**3, VAR-SPH1, RSQR(3)
      ELSE
         WRITE(11,905)
         WRITE(11,915) BETA(1,1), BETA(1,2)
         WRITE(11,925) BETA(2,1), BETA(2,2)
         WRITE(11,930) SPH1, SPH2
         WRITE(11,945) SPH2**3, VAR-SPH1
      ENDIF
      IF(NPTS.LE.10) THEN
         WRITE(950)
         DO 10 I=1,NPTS
            WRITE(11,960)Y(I),YHAT(1,I),YHAT(2,I),YHAT(3,I)
 10      CONTINUE
      ENDIF
  900 FORMAT(//,31X,'MODEL',30X,'R-SQUARED')
  905 FORMAT(//,31X,'MODEL')
  910 FORMAT(1X,'LINEAR',12X,'Y(H) = ',F6.3,' + ',F6.3,' * H',22X,F6.4)
  915 FORMAT(1X,'LINEAR',12X,'Y(H) = ',F6.3,' + ',F6.3,' * H')
  920 FORMAT(1X,'DE WIJSIAN',8X,'Y(H) = ',F6.3,' + ',F6.3,' * LN(H)',
     &18X,F6.4)
  925 FORMAT(1X,'DE WIJSIAN',8X,'Y(H) = ',F6.3,' + ',F6.3,' * LN(H)')
  930 FORMAT(1X,'SPHERICAL',9X,'Y(H) = ',F6.3,' * [ 1.5 * H / ',F6.3,
     &' - .5 *')
  940 FORMAT(27X,'H^3 / ',F9.3,' ] + ',F6.3,14X,F6.4)
  945 FORMAT(27X,'H^3 / ',F9.3,' ] + ',F6.3)
  950 FORMAT(///,3X,'ACTUAL',9X,'LINEAR',7X,'DE WIJSIAN',5X,'SPHERICAL')
  960 FORMAT(1X,4(F10.3,5X))
      RETURN
      END
```

Variogram Calculation Program

The variogram calculation program reads from the grid files produced with either the FORTRAN or C GRID programs. The output file consists of h, $\gamma(h)$, and the number of points used in determining $\gamma(h)$. A file for input to the IDL variogram procedure is also generated.

The code is provided below.

```
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C  THIS PROGRAM WAS USED TO DETERMINE THE VARIOGRAMS FOR THE FINAL  C
C  FACES.  THE RESIDUALS ARE LOCATED IN [MGRANT.RESIDUALS].  THE    C
C  OUTPUT IS LOCATED IN [MGRANT.VARIOGRAMS].  THIS FILE READS FROM  C
C  VARIO.SET AND THE RESIDUAL FILE.  THE GRID CONTAINS VALUES IN A  C
C  100 BY 50 ARRAY.          13 JAN 90                              C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
      COMMON RVALUE(200,200),TEMP(2,400,400),NPTS(400),MAX,MDIR,MLAG,
     1NOFH,GAMMA,NROW,NCOL,NDIAG,AVGR,A(4),ISTAT,IDLFLAG
      CALL INITL
      DO 10 IDIR=1,MDIR
         CALL FORM(IDIR)
         DO 10 ILAG=1,MLAG
            CALL VAR(ILAG)
            CALL OUTPUT(IDIR,ILAG)
   10    CONTINUE
   20 CONTINUE
      END

      SUBROUTINE INITL
      COMMON RVALUE(200,200),TEMP(2,400,400),NPTS(400),MAX,MDIR,MLAG,
     1NOFH,GAMMA,NROW,NCOL,NDIAG,AVGR,A(4),ISTAT,IDLFLAG
      CHARACTER FILIN*32,FILOUT*32,ANSWER*3,IDLOUT*32
      WRITE(*,*) 'INPUT THE NAME OF THE INPUT FILE'
      READ(*,'(A32)') FILIN
      WRITE(*,*) 'INPUT THE NAME OF THE OUTPUT FILE'
```

```
      READ(*,'(A32)') FILOUT
      WRITE(*,*) 'DO YOU WANT TO CREATE AN IDL FILE?'
      READ(*,'(A3)') ANSWER
      IF (ANSWER.EQ.'YES') THEN
         WRITE(*,*) 'INPUT THE NAME OF THE IDL FILE'
         READ(*,'(A32)') IDLOUT
         OPEN(12,FILE=IDLOUT,STATUS='NEW')
         IDLFLAG = 1
      ENDIF
      OPEN(10,FILE=FILIN,STATUS='OLD')
      OPEN(11,FILE=FILOUT,STATUS='NEW')
      OPEN(9,FILE='VARIO.SET',STATUS='OLD')
C      MAX - INDICATES THE MAXIMUM NUMBER OFDATA POINTS
C      MDIR - THE NUMBER OF DIRECTIONS
C      MLAG - THE NUMBER OF LAGS
C      NROW - THE NUMBER OF ROWS
C      NCOL - THE NUMBER OF COLUMNS
C      ISTAT - =1 IF AVERAGE RADIUS IS TO BE CALCULATED
      READ(9,*)MAX,NROW,NCOL,MDIR,MLAG,ISTAT
      READ(9,*)A(1),A(2),A(3),A(4)
      IF(NROW.GT.NCOL) THEN
         NDIAG=NROW
      ELSE
         NDIAG=NCOL
      ENDIF
      IF(ISTAT.EQ.1) THEN
         TOTAL=0
         PTS=0
      ENDIF
      DO 20 I=1,NROW
         DO 10 J=1,NCOL
            RVALUE(I,J)=0
   10    CONTINUE
   20 CONTINUE
      DO 50 I=1,NROW
         DO 40 J=0,NCOL-10,10
         READ(10,*,END=60)(RVALUE(I,J+K),K=1,10)
         IF(ISTAT.EQ.1) THEN
            DO 30 L=1,10
               IF(RVALUE(I,L+J).NE.0) THEN
                  TOTAL=TOTAL+RVALUE(I,J+L)
                  PTS=PTS+1
```

```
            ENDIF
 30       CONTINUE
        ENDIF
 40 CONTINUE
 50 CONTINUE
 60 CONTINUE
    IF(ISTAT.EQ.1) THEN
       AVGR=TOTAL/PTS
    ENDIF
    RETURN
    END

    SUBROUTINE FORM(IDIR)
    COMMON RVALUE(200,200),TEMP(2,400,400),NPTS(400),MAX,MDIR,MLAG,
   1NOFH,GAMMA,NROW,NCOL,NDIAG,AVGR,A(4),ISTAT,IDLFLAG
    DO 30 I=1,2
       DO 20 J=1,NROW+NCOL-1
          DO 10 K=1,NDIAG
             TEMP(I,J,K)=0
 10       CONTINUE
 20    CONTINUE
 30 CONTINUE
    GO TO (100,200,300,400),IDIR
100 CONTINUE
    DO 120 IROW=1,NROW
       NCNT=0
       DO 110 ICOL=1,NCOL
          IF(RVALUE(IROW,ICOL).NE.0) THEN
             NCNT=NCNT+1
             TEMP(1,IROW,NCNT)=RVALUE(IROW,ICOL)
             TEMP(2,IROW,NCNT)=ICOL
          ENDIF
110    CONTINUE
       NPTS(IROW)=NCNT
120 CONTINUE
    RETURN
200 CONTINUE
    DO 220 ICOL=1,NCOL
       NCNT=0
       DO 210 IROW=1,NROW
          IF(RVALUE(IROW,ICOL).NE.0) THEN
             NCNT=NCNT+1
```

```
            TEMP(1,ICOL,NCNT)=RVALUE(IROW,ICOL)
            TEMP(2,ICOL,NCNT)=IROW
         ENDIF
210     CONTINUE
        NPTS(ICOL)=NCNT
220  CONTINUE
     RETURN
300  CONTINUE
     DO 320 IROW=1,NROW
        IF(IROW.LE.NCOL) THEN
           INDX=IROW
        ELSE
           INDX=NCOL
        ENDIF
        NCNT=0
        DO 310 I=1,INDX
           IF(RVALUE(IROW+1-I,I).NE.0) THEN
              NCNT=NCNT+1
              TEMP(1,IROW,NCNT)=RVALUE(IROW+1-I,I)
              TEMP(2,IROW,NCNT)=I
           ENDIF
310     CONTINUE
        NPTS(IROW)=NCNT
320  CONTINUE
     DO 340 ICOL=2,NCOL
        IF(NCOL-ICOL.LT.NROW) THEN
           INDX=NCOL-ICOL+1
        ELSE
           INDX=NROW
     ENDIF
        NCNT=0
        DO 330 I=1,INDX
            IF(RVALUE(NROW+1-I,ICOL-1+I).NE.0) THEN
               NCNT=NCNT+1
               TEMP(1,NROW+ICOL-1,NCNT)=RVALUE(NROW+1-I,ICOL-1+I)
               TEMP(2,NROW+ICOL-1,NCNT)=I
            ENDIF
330     CONTINUE
        NPTS(NROW+ICOL-1)=NCNT
340  CONTINUE
     RETURN
400  CONTINUE
```

```
      DO 420 IROW=1,NROW
         IF(IROW.LE.NCOL) THEN
            INDX=IROW
         ELSE
            INDX=NCOL
         ENDIF
         NCNT=0
         DO 410 I=1,INDX
            IF(RVALUE(NROW+I-IROW,I).NE.0) THEN
               NCNT=NCNT+1
               TEMP(1,IROW,NCNT)=RVALUE(NROW+I-IROW,I)
               TEMP(2,IROW,NCNT)=I
            ENDIF
 410     CONTINUE
         NPTS(IROW)=NCNT
 420  CONTINUE
      DO 440 ICOL=2,NCOL
         IF(NCOL-ICOL.LT.NROW) THEN
            INDX=NCOL-ICOL+1
         ELSE
            INDX=NROW
         ENDIF
         NCNT=0
         DO 430 I=1,INDX
            IF(RVALUE(I,ICOL-1+I).NE.0) THEN
               NCNT=NCNT+1
               TEMP(1,NROW+ICOL-1,NCNT)=RVALUE(I,ICOL-1+I)
               TEMP(2,NROW+ICOL-1,NCNT)=I
            ENDIF
 430     CONTINUE
         NPTS(NROW+ICOL-1)=NCNT
 440  CONTINUE
      RETURN
      END

      SUBROUTINE VAR(ILAG)
      COMMON RVALUE(200,200),TEMP(2,400,400),NPTS(400),MAX,MDIR,MLAG,
     1NOFH,GAMMA,NROW,NCOL,NDIAG,AVGR,A(4),ISTAT,IDLFLAG
      NOFH=0
      SUM=0
      DO 30 INDX=1,NROW+NCOL-1
         DO 20 I=1,NPTS(INDX)-1
```

```
         DO 10 J=1,ILAG
            IF(I+J.LE.NPTS(INDX)) THEN
               IF (TEMP(2,INDX,I+J)-TEMP(2,INDX,I).EQ.ILAG) THEN
                  NOFH=NOFH+1
                  SUM=SUM+(TEMP(1,INDX,I+J)-TEMP(1,INDX,I))**2
               ENDIF
            ENDIF
 10      CONTINUE
 20   CONTINUE
 30 CONTINUE
    IF(NOFH.GT.0.AND.SUM.GT.0) THEN
      GAMMA=1.0/(2*NOFH)*SUM
    ELSE
      GAMMA=0.0
    ENDIF
    RETURN
    END

    SUBROUTINE OUTPUT(IDIR,ILAG)
    COMMON RVALUE(200,200),TEMP(2,400,400),NPTS(400),MAX,MDIR,MLAG,
   1NOFH,GAMMA,NROW,NCOL,NDIAG,AVGR,A(4),ISTAT,IDLFLAG
    IF(IDIR.EQ.1.AND.ILAG.EQ.1) THEN
       IF(ISTAT.EQ.1) THEN
          WRITE(11,*)'AVERAGE RADIUS = ',AVGR
          WRITE(11,*)
       ENDIF
       WRITE(11,*)'DIRECTION   LAG         A          GAMMA    NPTS '
    ENDIF
    WRITE(11,900)IDIR,ILAG,ILAG*A(IDIR),GAMMA,NOFH
    IF (IDLFLAG.EQ.1) THEN
       WRITE(12,910)ILAG*A(IDIR),GAMMA,NOFH
    ENDIF
900 FORMAT(5X,I1,7X,I3,5X,F7.2,6X,F7.2,4X,I4)
910 FORMAT(F7.2,6X,F7.2,6X,I5)
    RETURN
    END
```

C Program for Removing Trend

This program reads in the output of the coordinate labelling program for both the subject and the average face. The data is stored in two arrays and differenced to obtain an array of residuals. The output is similar to the two input files in that it follows the same grid structure. This output file is used as input to the plotting routines provided in Appendix J.

The code is provided below.

```
/*
     This program reads in the output of fill.c  for both the mean face
     and a subject into two NROW by NCOL arrays.  The mean values are then
     subtracted from the subject values to produce a grid of residuals
     which can be used for plotting or as data for the variogram programs.

     To run this program, type:

      differ.x <subject fn> <mean face fn> <output fn>

*/

#include <stdio.h>
#include<math.h>

#define NROW 100
#define NCOL 50

main(argc,argv)
     int argc;
     char *argv[];

{
     int i,j,k,irow,icol,NUM;
     float x,y,z,d1,d2,
     float Grid[NROW][NCOL];
```

```
float Mean_Face[NROW][NCOL];
float Subject[NROW][NCOL];

FILE *f_face, *f_mean, *fout;

f_face = fopen(argv[1],"r");
f_mean = fopen(argv[2],"r");
fout = fopen(argv[3],"w");

NUM = NROW * NCOL;

for(i=0; i<NROW; i++) {
   for(j=0; j<NCOL; j++) {
      Grid[i][j] = 0.0;
      Mean_Face[i][j] = 0.0;
      Subject[i][j] = 0.0;
   }
}

for(i=0; i<NUM; i++) {
   fscanf(f_face,"%d %d %f %f %f %f %f\n",&irow,&icol,&x,&y,&z,&d1,&d2);
   Subject[irow][icol] = z;
}

for(i=0; i<NUM; i++) {
   fscanf(f_mean,"%d %d %f %f %f %f %f\n",&irow,&icol,&x,&y,&z,&d1,&d2);
   Mean_Face[irow][icol] = z;
}

for(i=0; i<NROW; i++) {
   for(j=0; j<NCOL; j++) {
      if((Mean_Face[i][j]!=0)&&(Subject[i][j]!=0)) {
         Grid[i][j] = Subject[i][j] - Mean_Face[i][j];
      }
   }
}
for(i=0; i<NROW; i++) {
   for(j=0; j<=NCOL-10; j=j+10) {
      for(k=0; k<10; k++) {
         fprintf(fout,"%f ",Grid[i][j+k]);
      }
      fprintf(fout,"\n");
```

```
        }
    }
    fclose(f_face);
    fclose(f_mean);
    fclose(fout);
}
```

FORTRAN Program for Removing Trend

This program is a FORTRAN version of the preceeding program. The input and output files are identical to the files described in the previous section.

The code is as follows.

```
      PROGRAM MAIN
      DIMENSION GRID(200,200), FACE(200,200), RMEAN(200,200)
      CHARACTER INFILE1*32, INFILE2*32, OUTFILE*32
      WRITE(6 *) ' THIS PROGRAM FORMS A GRID OF RESIDUALS FOR EACH FACE'
      WRITE(6,*) ' '
      WRITE(6,*) ' ENTER THE NAME OF THE INPUT FILE FOR THE FACE'
      READ(5,'(A32)') INFILE1
      WRITE(6,*) 'ENTER THE NAME OF THE INPUT FILE FOR THE MEAN'
      READ(5,'(A32)') INFILE2
      WRITE(6,*) ' ENTER THE NAME OF THE OUTPUT FILE'
      READ(5,'(A32)') OUTFILE
      WRITE(6,*) 'ENTER THE GRID DIMENSIONS [NROW NCOL]'
      READ(5,*) NROW, NCOL
      OPEN(10,FILE=INFILE1,STATUS='OLD')
      OPEN(11,FILE=INFILE2,STATUS='OLD')
      OPEN(12,FILE=OUTFILE,STATUS='NEW')
      NUM=NROW*NCOL
      DO 20 I=1,NROW
         DO 10 J=1,NCOL
            GRID(I,J)=0.0
            FACE(I,J)=0.0
            RMEAN(I,J)=0.0
   10    CONTINUE
   20 CONTINUE
      DO 30 I=1,NUM
         READ(10,*,END=40) IROW,ICOL,X,Y,Z
         FACE(IROW+1,ICOL+1)=Z
   30 CONTINUE
   40 CONTINUE
      DO 50 I=1,NUM
```

```
      READ(11,*,END=60) IROW,ICOL,X,Y,Z
      RMEAN(IROW+1,ICOL+1)=Z
 50 CONTINUE
 60 CONTINUE
    DO 80 I=1,NROW
      DO 70 J=1,NCOL
        IF(FACE(I,J).NE.0) GRID(I,J)=FACE(I,J)-RMEAN(I,J)
 70 CONTINUE
 80 CONTINUE
    DO 110 I=1,NROW
      DO 100 J=0,NCOL-10,10
        WRITE(12,900)(GRID(I,J+K),K=1,10)
100   CONTINUE
110 CONTINUE
900 FORMAT(10(F7.3,2X))
    END
```

C Program for Forming Grid

This program structures the output of the coordinate labelling program into a grid format. The input file is the output of the coordinate labelling program. The code is provided below.

```
/*
     This program reads in the output of fill.c into an NROW by NCOL
     array which is then output to a file for plotting or updating.

     To run the program, type:

     grid.x <subject fn> <output fn>

*/

#include <stdio.h>
#include<math.h>

#define NROW 100
#define NCOL 50

main(argc,argv)
     int argc;
     char *argv[];

{
     int i,j,k,irow,icol,NUM;
     float x,y,z,d1,d2;
     float Grid[NROW][NCOL];

     FILE *fin, *fout;

     fin = fopen(argv[1],"r");
     fout = fopen(argv[2],"w");

     NUM = NROW * NCOL;
```

```
    for(i=0; i<NROW; i++) {
       for(j=0; j<NCOL; j++) {
          Grid[i][j] = 0.0;
       }
    }

    for(i=0; i<NUM; i++) {
       fscanf(fin,"%d %d %f %f %f %f %f\n",&irow,&icol,&x,&y,&z,&d1,&d2);
       Grid[irow][icol] = z;
    }

    for(i=0; i<NROW; i++) {
       for(j=0; j<=NCOL-10; j=j+10) {
          for(k=0; k<10; k++) {
             fprintf(fout,"%f ",Grid[i][j+k]);
          }
      fprintf(fout,"\n");
       }
    }
    fclose(fin);
    fclose(fout);
}
```

FORTRAN Program for Forming Grid

This is the FORTRAN version of the preceeding program.

```
      PROGRAM MAIN
      DIMENSION GRID(200,200)
      CHARACTER INFILE*32, OUTFILE*32
      WRITE(6,*) ' THIS PROGRAM FORMS A GRID FOR THE FACES'
      WRITE(6,*) ' '
      WRITE(6,*) ' ENTER THE NAME OF THE INPUT FILE'
      READ(5,'(A32)') INFILE
      WRITE(6,*) ' ENTER THE NAME OF THE OUTPUT FILE'
      READ(5,'(A32)') OUTFILE
      WRITE(6,*) 'ENTER THE GRID DIMENSIONS [NROW NCOL]'
      READ(5,*) NROW, NCOL
      OPEN(10,FILE=INFILE,STATUS='OLD')
      OPEN(11,FILE=OUTFILE,STATUS='NEW')
      NUM=NROW*NCOL
      DO 20 I=1,NROW
         DO 10 J=1,NCOL
            GRID(I,J)=0.0
 10      CONTINUE
 20   CONTINUE
      DO 30 I=1,NUM
         READ(10,*,END=99) IROW,ICOL,X,Y,Z
         GRID(IROW+1,ICOL+1)=Z
 30   CONTINUE
 99   CONTINUE
      DO 110 I=1,NROW
         DO 100 J=0,NCOL-10,10
            WRITE(11,900)(GRID(I,J+K),K=1,10)
100      CONTINUE
110   CONTINUE
900   FORMAT(10(F7.3,2X))
      END
```

C Program for Consolidating Variogram Files

This program consolidates the output of the experimental variogram program. The files to be combined are specified in a list.X file which is passed as an argument in the execution command. The code for this program is as follows.

```
/*
     This program combines the variogram data for a specified number
     of subjects and directions in the proper format for input into
     the model fitting program.

     To run this program, type:

     con.x <filename>

*/

#include <stdio.h>
#include <math.h>


/*
     NPTS: the number of points in each variogram file
     MAXSUBS: the maximum number of subject files
     VPTS: the number of points in each direction of a file
*/

#define NPTS 40
#define MAXSUBS 30
#define VPTS 10


main(argc,argv)
     int argc;
     char *argv[];

/*
```

```
    i,j: loop variables
    nosub: number of subject files in the input file
    h[][]: the distance
    g[][]: gamma(h)
    n[][]: the number of points used in the gamma(h) calculation
    subject_name[]: temporary variable for subject file names
*/

{
    int i,j;
    int nosub;
    float h[MAXSUBS][NPTS];
    float g[MAXSUBS][NPTS];
    float n[MAXSUBS][NPTS];
    char subject_name[10];
    float t;


    FILE *fin, *fdata, *fd1, *fd2, *fd3, *fd4, *fall;

    fin = fopen(argv[1],"r");
    fd1 = fopen("dir1.out","w");
    fd2 = fopen("dir2.out","w");
    fd3 = fopen("dir3.out","w");
    fd4 = fopen("dir4.out","w");
    fall = fopen("dirall.out","w");

    fscanf(fin,"%d\n",&nosub);

    for(i=0; i<nosub; i++) {
     fscanf(fin,"%s\n",subject_name);
         printf("%s\n",subject_name);
     fdata = fopen(subject_name,"r");
     for(j=0; j<NPTS; j++) {
          fscanf(fdata,"%f %f %f\n",&h[i][j],&g[i][j],&n[i][j]);
         }
    }

    for(i=0; i<VPTS; i++) {
      for(j=0; j<nosub; j++) {
        fprintf(fd1,"%f %f %f\n",h[j][i],g[j][i],n[j][i]);
        fprintf(fd2,"%f %f %f\n",h[j][i+VPTS],g[j][i+VPTS],n[j][i+VPTS]);
```

```
        fprintf(fd3,"%f %f %f\n",h[j][i+VPTS*2],g[j][i+VPTS*2],n[j][i+VPTS*2]);
        fprintf(fd4,"%f %f %f\n",h[j][i+VPTS*3],g[j][i+VPTS*3],n[j][i+VPTS*3]);
      }
    }

    for(i=0; i<NPTS; i++) {
      for(j=0; j<nosub; j++) {
        fprintf(fall,"%f %f %f\n",h[j][i],g[j][i],n[j][i]);
      }
    }
}
```

FORTRAN Program for Consolidating Variogram Files

This program is similar to the preceeding program. However, a list file is not used. The files must be specified in the FORTRAN code.

```
      PROGRAM MAIN
      DIMENSION X(5,100),Y(5,100),N(5,100)
      OPEN(10,FILE='S1.DAT',STATUS='OLD')
      OPEN(11,FILE='S10.DAT',STATUS='OLD')
      OPEN(12,FILE='S118.DAT',STATUS='OLD')
      OPEN(13,FILE='S122.DAT',STATUS='OLD')
      OPEN(14,FILE='S186.DAT',STATUS='OLD')
      OPEN(15,FILE='D1.OUT',STATUS='NEW')
      OPEN(16,FILE='D2.OUT',STATUS='NEW')
      OPEN(17,FILE='D3.OUT',STATUS='NEW')
      OPEN(18,FILE='D4.OUT',STATUS='NEW')
      OPEN(19,FILE='ALL.OUT',STATUS='NEW')
      DO 10 I=1,40
         READ(10,*)X(1,I),Y(1,I),N(1,I)
   10 CONTINUE
      DO 20 I=1,40
         READ(11,*)X(2,I),Y(2,I),N(2,I)
   20 CONTINUE
      DO 30 I=1,40
         READ(12,*)X(3,I),Y(3,I),N(3,I)
   30 CONTINUE
      DO 40 I=1,40
         READ(13,*)X(4,I),Y(4,I),N(4,I)
   40 CONTINUE
      DO 50 I=1,40
         READ(14,*)X(5,I),Y(5,I),N(5,I)
   50 CONTINUE
      DO 70 I=1,10
         DO 60 J=1,5
            WRITE(15,*)X(J,I),Y(J,I),N(J,I)
            WRITE(16,*)X(J,I+10),Y(J,I+10),N(J,I+10)
            WRITE(17,*)X(J,I+20),Y(J,I+20),N(J,I+20)
```

```
         WRITE(18,*)X(J,I+30),Y(J,I+30),N(I,I+30)
60    CONTINUE
70 CONTINUE
   DO 90 I=1,40
      DO 80 J=1,5
         WRITE(19,*)X(J,I),Y(J,I),N(J,I)
80    CONTINUE
90 CONTINUE
   END
```

Appendix H. *Kriging Programs*

The appendix includes the kriging programs. The first program is the C program for kriging the residuals. The second program verifies the results of the first program and corrects for numerical difficulties. The last program combines the kriged surfaces with the trend which was removed during the kriging process. The first several comment lines f each program provide user instructions. The code may be obtained by contacting Major Robinson.

C Program for Kriging

```
/*
     This program determines the estimate and the estimation variance
     for the value at each mid point of the grid for each data set.
     Currently, the program performs universal kriging with the
     x and y terms present for the linear drift.  This program reads
     in the data set values, adjusts for the trend by subtracting the
     mean values for the sample faces, and kriges the residuals to
     obtain the estimates and the variances.

     To compile the program, type:

      cc -o krige.x krige.c -lm

     To run the program, type:

      krige.x <list fn> <mean fn> <surface output fn> <var output fn>

     (Please reference the user's manual for more details.)


     Date of last modification: 22 JAN 90
     Date of last modification: 25 JAN 90 dgr
*/

#include <stdio.h>
#include <math.h>
#include "nr.h"
#include "nrutil.h"

/*
     The following are the defined parameters.

     NX,NY: number of increments on the x,y axes
     XMAX,YMAX: maximum values for x,y
     XMIN,YMIN: minimum values for x,y
     DX,DY: the length of the increments for the x,y axes
     KX,KY: scale parameterss for x,y axes
     IA: the integer value which defines the zone of influence
     A: the zone of influence (range)
     C: the sill minus the nugget effect (sill - C0)
     C0: the nugget effect
```

```
     MAXKPTS: the maximum number of known points in a zone

      [Note:  IA and MAXKPTS are related in that if IA is
      large, kpts could possibly exceed the value of
      MAXKPTS.  The trade-off is between estimation error
      and computational efficiency.  Reference documentation
      for more information.]
*/

#define NX 100
#define NY 50
#define XMAX 4.
#define XMIN 0.
#define YMAX 300.
#define YMIN 100.
#define DX (XMAX-XMIN)/NX
#define DY (YMAX-YMIN)/NY
#define KX (1.0)/(DX)
#define KY (1.0)/(DY)
#define IA 7
#define A 6.645
#define C 2.226
#define C0 0.689
#define MAXKPTS 200

/*
     The following are global variables.

     kpts: the number of points in the gamma structure
     delta: the value to decrement IA if kpts > MAXKPTS
     Mean_Face[][]: the array of means for the mean face
     Sample[][]: the known points in a zone
     MatA[][]: the A matrix in the AX=B format for the kriging system
     MatB[]: the B matrix in the AX=B format for the kriging system
     MatX[]: the X matrix in the AX=B format for the kriging system
     Grid[][][]: the array of  kriged estimates and variances
*/

int kpts,delta;
float Sample[MAXKPTS][3];
float Mean_Face[NX][NY];
float MatA[MAXKPTS+3][MAXKPTS+3];
```

```
float MatB[MAXKPTS+3];
float MatX[MAXKPTS+3];
float Grid[2][NX][NY];

     /*
      The dimension for the Mat* matrices allows for 3 more
      terms than the number of points in a sample:
         1 - one for the row and column of ones,
         2 - one for the x term for the linear drift, and
         3 - one for the y term for the linear drift.
     */

struct point {
     float angle, latitude, radius;
     struct point *next;
};

struct point *point_array[NX][NY];

void Krerror();

main(argc,argv)
     int argc;
     char *argv[];

/*
     This is the main portion of the program.

     The following are local variables.

     i,j: loop variables
*/

{
     int i,j;
     int iii;

     void Data_In();
     void Mean_In();
     void Def_Zone();
     void Find_Pts();
     void Build_A();
```

```
    void Build_B();
    void Build_X();
    void Estimate();
    void Output();
    void InputCheck();

    InputCheck(argc, argv);

/*    FILE *flog;
    flog = fopen("Sample.out","w"); */

/* printf("%f %f %d %f %f %f %f %f\n",DX,DY,IA,A,C,CO,KX,KY); */

    Data_In(argv[1]);
    Mean_In(argv[2]);

    for(i=0; i<NX; i++) {
       for(j=0; j<NY; j++) {
/*  printf("%d %d\n",i,j);*/
           if(Mean_Face[i][j]!=0.0) {
          kpts = 0;
          delta = 0;
          do {
             Def_Zone(i,j);
/*  printf("%d %d %d %d\n",i,j,kpts,delta); */
             delta++;
             } while ((kpts>=MAXKPTS)&&(delta<=IA));
          if(delta>IA+1) {
                printf("Warning: grid size insufficient.\n"); }

          if(kpts>0) {

 /* for(iii=0; iii<kpts; iii++) {
 fprintf(flog,"%f %f %f\n",Sample[iii][0],Sample[iii][1],Sample[iii][2]);
  } */

                   Build_A();
             Build_B(i,j);
             Build_X();
             Estimate(i,j);
              }
           }
```

```
    }
     }
     Output(argv[3],argv[4]);
 /*    fclose(flog); */
}




void Data_In(arg1)
char *arg1;

/*
     This routine reads in the data.  This routine was adapted from
     Dr. David G. Robinson's fill.c program.

     The following are local variables.

     i,j,k: loop variables
     m,n: block labels
     npts: number of data points for a single data set
     nosub: number of subject data files to be read
     x: angle
     y: altitude
     z: radius
     d1,d2,d3: dummy variables
     subject_name: name of the subject data file to be read
*/

{
     int i,j,k,m,n;
     int npts, nosub;
     float x,y,z;
     float d1, d2, d3;
     char subject_name[10];

     FILE *fins, *fin;

     struct point *temp;

     /* initializing pointers to NULL  */
```

```
    for (i=0; i<NX; i++)
         for(j=0; j<NY; j++)
          point_array[i][j] = NULL;

    fins = fopen(arg1,"r");
    fscanf(fins,"%d\n",&nosub);

    for(k=0;k<nosub;k++) {
         fscanf(fins,"%s %d\n",subject_name, &npts);
         fin = fopen(subject_name,"r");

         for (i=0; i<npts; i++) {
           fscanf(fin,"%f %f %f %f %f %f\n", &x, &y, &z, &d1, &d2, &d3);

              m = (int)(NY*(y-YMIN)/(YMAX-YMIN));
              n = (int)(NX*(x-XMIN)/(XMAX-XMIN));
              temp = (struct point *)malloc(sizeof(struct point));

              if((m>=0)&&(n>=0)){
               if (temp != NULL)  {
                temp->angle    = x;
                temp->latitude = y;
                temp->radius   = z;
                temp->next     = point_array[n][m];
                point_array[n][m] = temp;
               }
              }
         }
    fclose(fin);
    }
}



void Mean_In(arg2)
char *arg2;

/*
     This routine reads in the means for each grid point from
```

```
    the mean face.  The mean face is the average value of each
    (i,j) grid point for the sample of 30 faces.

    The following are local variables.

    i,j: loop variables
*/

{
    int i,j;

    FILE *f_face;

    f_face = fopen(arg2,"r");

    for(i=0; i<NX; i++) {
       for(j=0; j<NY/10; j++) {

      fscanf(f_face,"%f %f %f %f %f %f %f %f %f %f\n",&Mean_Face[i][j*10],
      &Mean_Face[i][j*10+1],&Mean_Face[i][j*10+2],&Mean_Face[i][j*10+3],
      &Mean_Face[i][j*10+4],&Mean_Face[i][j*10+5],&Mean_Face[i][j*10+6],
      &Mean_Face[i][j*10+7],&Mean_Face[i][j*10+8],&Mean_Face[i][j*10+9]);

       }
    }
    fclose(f_face);
}



void Def_Zone(i,j)
int i,j;

/*
    This routine determines which blocks are included in the zone
    and calls the Find_Pts routine to find the points in all
    of the blocks.

    The following are local variables.
```

```
     ii,jj: loop variables
     ilow,jlow: lower end of the zone
     ihigh,jhigh: upper end of the zone
*/

{
     int ii, ilow, ihigh;
     int jj, jlow, jhigh;

     ilow = i - (IA-delta);
     jlow = j - (IA-delta);
     ihigh = i + (IA-delta);
     jhigh = j + (IA-delta);

     if(i-(IA-delta)<0) ilow = 0;
     if(j-(IA-delta)<0) jlow = 0;
     if(i+(IA-delta)>NX) ihigh = NX;
     if(j+(IA-delta)>NY) jhigh = NY;

     kpts=0;

     for(ii=ilow; ii<=ihigh; ii++) {
      for(jj=jlow; jj<=jhigh; jj++) {
           if(Mean_Face[ii][jj]!=0.0) {
                   Find_Pts(ii,jj,point_array[ii][jj]);
           }
      }
     }
}



void Find_Pts(ii,jj,ps)
int ii,jj;
struct point *ps;

/*
     This routine fills the Sample array with the points within a
     zone.  The mean values (Mean_Face[][]) are subtracted from the
     data to produce the residual values.
```

```
     The following are local variables.

     x: x coordinate of the (i,j) block midpoint
     y: y coordinate of the (i,j) block midpoint
     h: the distance between any two points
     temp1,temp2: temporary variables
*/

{
     float x,y,h,temp1,temp2;


     x = ((ii+.5)*DX) + XMIN;
     y = ((jj+.5)*DY) + YMIN;

     if((ps!=NULL)&&(kpts<MAXKPTS)) {
        do
        {
            temp1 = (x-(ps->angle));
            temp2 = (y-(ps->angle));
            h = sqrt(KX*KX*temp1*temp1+KY*KY*temp2*temp2);

    if(h>0.0) {

           Sample[kpts][0] = (ps->angle);
           Sample[kpts][1] = (ps->latitude);

           if((ps->radius)!=0) {
                Sample[kpts][2] = (ps->radius) - Mean_Face[ii][jj];
           }
           else {
                Sample[kpts][2] = (ps->radius);
           }

           /*
                If the value of radius = 0, the mean is not subtracted
                because subtracting the mean would provide a large
                residual which would then be kriged and readded to
                the mean resulting in a value almost double what it
                should be.
           */
```

```
            kpts++;
          }
        else {
          printf("Note: A sample point coincides with a grid point\n");
        }
          ps = ps->next;
        } while ((ps!=NULL)&&(kpts<MAXKPTS));
        }
}




void Build_A()

/*
     This routine builds the A matrix of the kriging equations.

     The following are local variables.

     i,j: loop variables
     temp1,temp2,gamma: temporary variables
     h: the distance between any two points
*/

{
     int i,j;
     double h,temp1,temp2,gamma;

     /*
      This portion completes the gamma structure of A.
     */

/* printf("%f %f\n",DX,DY); */

     for(i=0; i<kpts; i++) {
      MatA[i][i]=0.0;
      for(j=0; j<kpts; j++) {
           temp1 = (Sample[i][0]-Sample[j][0]);
           temp2 = (Sample[i][1]-Sample[j][1]);
```

```
            h = sqrt(KX*KX*temp1*temp1+KY*KY*temp2*temp2);
            temp1 = h*h*h;
            temp2 = A*A*A;
            if(h!=0) {
               if(h<A){
                    gamma = C * (1.5*h/A - 0.5*temp1/temp2) + C0;
                    MatA[i][j]= gamma;
                }
                else { MatA[i][j]= C + C0;
                }
            }
            else { MatA[i][j] = 0.0;
            }
            MatA[j][i]=MatA[i][j];
 /* printf("%d %d %f %f\n",i,j,h,MatA[i][j]); */
       }
      }

      /*
       This portion adds a column and row of 1's.
      */

      for(i=0; i<kpts; i++) {
       MatA[i][kpts] = 1.0;
/* printf("%d %d %f\n",i,kpts,MatA[i][kpts]); */
       MatA[kpts][i] = 1.0;
/* printf("%d %d %f\n",kpts,i,MatA[kpts][i]); */
      }

      /*
       This portion provides terms for a linear trend.
      */

      for(i=0; i<kpts; i++) {
       for(j=0; j<2; j++) {
            MatA[i][kpts+1+j]=Sample[i][j];
/* printf("%d %d %f\n",i,1+j+kpts,MatA[i][kpts+1+j]); */
            MatA[kpts+j+1][i]=Sample[i][j];
/* printf("%d %d %f\n",j+kpts+1,i,MatA[1+kpts+j][i]); */
           }
      }
```

```
    /*
     This portion completes A with a block of 0's.
    */

    for(i=0; i<3; i++) {
     for(j=0; j<3; j++) {
          MatA[kpts+i][kpts+j] = 0.0;
/* printf("%d %d %f\n",i+kpts,j+kpts,MatA[kpts+i][kpts+j]); */
         }
    }
}



void Build_B(i,j)
int i,j;

/*
    This routine builds the B matrix of the kriging equations.

    The following are local variables.

    ii: loop variable
    x: x coordinate of the (i,j) block midpoint
    y: y coordinate of the (i,j) block midpoint
    h: the distance between any two points
    temp1,temp2,gamma: temporary variables
*/

{
    int ii;
    float x,y;
    double h,temp1,temp2,gamma;
    x = ((i+.5)*DX) + XMIN;
    y = ((j+.5)*DY) + YMIN;


/* printf("%f %f\n",DX,DY); */
/* printf("%d %d %f %f %f\n",i,j,x,y,A); */
```

```
    for(ii=0; ii<kpts; ii++) {
/* printf("%d %d %f %f %d %f %f\n",i,j,x,y,ii,Sample[ii][0],Sample[ii][1]);*/
          temp1 = (x-Sample[ii][0]);
          temp2 = (y-Sample[ii][1]);
          h = sqrt(KX*KX*temp1*temp1+KY*KY*temp2*temp2);
/* printf("%f %f %f\n",h,temp1,temp2);*/
          temp1 = h*h*h;
          temp2 = A*A*A;
          if(h!=0.0) {
               if(h<A){
                gamma = C * (1.5*h/A - 0.5*temp1/temp2) + C0;
                MatB[ii] = gamma;
/* printf("%d %f\n",ii,MatB[ii]); */
                }
                else { MatB[ii] = C + C0;
 /* printf("%d %f\n",ii,MatB[ii]); */
                }
           }
           else {
              MatB[ii] =0.0;
 /* printf("%d %f\n",ii,MatB[ii]); */
           }
      }

    MatB[kpts] = 1.0;
    MatB[kpts+1] = x;
    MatB[kpts+2] = y;

}




void Build_X()

/*
     This routine builds the X matrix of the kriging equations.

     The following variables are local variables.

     i,j: loop variables
```

```
    *indx, p, **a, *z, **c: "lu" variables
*/

{
    int i,j;
    int *indx;
    float p, **a, *z, **c;

    indx = ivector(1,kpts+3);
    a=matrix(1,kpts+3,1,kpts+3);
    z=vector(1,kpts+3);
    c=matrix(1,kpts+3,1,kpts+3);

/* printf("%d\n",kpts); */

    for(i=0; i<kpts+3; i++) {
        for(j=0; j<kpts+3; j++) {
            a[i+1][j+1] = MatA[i][j];
        }
/* printf("%f %f %f %f\n",a[i+1][1],a[i+1][2],a[i+1][3],a[i+1][4]); */
/* printf("%f %f %f\n",a[i+1][kpts+1],a[i+1][kpts+2],a[i+1][kpts+3]); */
    }

    ludcmp(a,kpts+3,indx,&p);

    for(i=0; i<kpts+3; i++) {
        z[i+1] = MatB[i];
    }

    lubksb(a,kpts+3,indx,z);

    for(i=0; i<kpts+3; i++) {
        MatX[i] = z[i+1];
    }
    free_matrix(a,1,kpts+3,1,kpts+3);
    free_matrix(c,1,kpts+3,1,kpts+3);
}
```

```
void Estimate(i,j)
int i,j;

/*
     This routine estimates the value at midpoint of the (i,j) block.

     The following are local variables.

     ii: loop variable
     sum, varsum: temporary variables
*/

{
     int ii;
     float sum, varsum;

     sum = 0.0;
     varsum = 0.0;

     for(ii=0; ii<kpts; ii++) {
        sum += MatX[ii] * Sample[ii][2];
        varsum += MatX[ii] * MatB[ii];
     }

     Grid[0][i][j] = sum;
     Grid[1][i][j] = varsum;

/* printf("%d %d %f %f\n",i,j,sum,varsum); */
/* printf("%f %f\n",sum,varsum); */

}




void Output(arg3,arg4)
char *arg3, *arg4;

/*
     This routine outputs the estimates and variances.
```

```
     The following are local variables.

     i,j,k: loop variable
*/

{
     int i,j,k;

     FILE *f_surf, *f_var, *f_dgr;

     f_surf = fopen(arg3,"w");
     f_var = fopen(arg4,"w");
/*     f_dgr = fopen("dgr.out","w"); */

     for (i=0; i<NX; i++) {
        for (j=0; j<=NY-10; j=j+10) {
           for(k=0; k<10; k++) {
              fprintf(f_surf,"%f ",Grid[0][i][j+k]);
           }
           fprintf(f_surf,"\n");

           for(k=0; k<10; k++) {
              fprintf(f_var,"%f ",Grid[1][i][j+k]);
           }
           fprintf(f_var,"\n");
        }
     }
     fclose(f_surf);
     fclose(f_var);
}

void InputCheck(argc, argv)
     int argc;
     char *argv[];

    {

    /* obtain file name from command string */
    /* check for correct number of arguments */

    if (argc !=3)
    {
```

```
    printf("\n%s requires file names as parameters\n",argv[0]);
    printf(" Usage: krige.x <list fn> <mean fn> <surface output fn>
     <var output fn> \n");
    printf(" Example:\n        krige.x list.a fmean.dat newmean.dat
    variance.dat\n");
    exit(0);
    }
    return;

/* argc is a counter from the command line, its the number of argv variables */
/* argv[0] contains the program name, it's a string */
}

/*******************************************************************/
/* Krerror:                                                        */
/*    allows gracefull exit from program in case of serious error */
/*******************************************************************/
void Krerror(error_text)
char error_text[];
{
    fprintf(stderr," Kriging run-time error...\n");
    fprintf(stderr,"%s\n",error_text);
    fprintf(stderr,"...now exiting to system...\n");
    exit(1);
}

/*
     The following routines were adapted from Numerical Recipes
     for C.
*/

/************************************************************/
#define TINY 1.0e-20;

void ludcmp(a,n,indx,d)
float **a;
int n,*indx;
float *d;
{
    int i,imax,j,k;
    float big,dum,sum,temp;
    float *vv,*vector();
```

```
	void nrerror(),free_vector();

	vv=vector(1,n);

	*d=1.0;
	for (i=1;i<=n;i++) {
		big=0.0;
		for (j=1;j<=n;j++)
			{
			if ((temp=fabs(a[i][j])) > big) big=temp;
			}
/*		if (big == 0.0) nrerror("Singular matrix in routine LUDCMP"); */
		if (big == 0.0) printf("Singular matrix in routine LUDCMP\n");
		vv[i]=1.0/big;
	}
	for (j=1;j<=n;j++) {
		for (i=1;i<j;i++) {
			sum=a[i][j];
			for (k=1;k<i;k++) sum -= a[i][k]*a[k][j];
			a[i][j]=sum;
		}
		big=0.0;
		for (i=j;i<=n;i++) {
			sum=a[i][j];
			for (k=1;k<j;k++)
				sum -= a[i][k]*a[k][j];
			a[i][j]=sum;
			if ( (dum=vv[i]*fabs(sum)) >= big) {
				big=dum;
				imax=i;
			}
		}
		if (j != imax) {
			for (k=1;k<=n;k++) {
				dum=a[imax][k];
				a[imax][k]=a[j][k];
				a[j][k]=dum;
			}
			*d = -(*d);
			vv[imax]=vv[j];
		}
		indx[j]=imax;
```

```
        if (a[j][j] == 0.0) a[j][j]=TINY;
        if (j != n) {
            dum=1.0/(a[j][j]);
            for (i=j+1;i<=n;i++) a[i][j] *= dum;
        }
    }
    free_vector(vv,1,n);
}

#undef TINY

void lubksb(a,n,indx,b)
int n,*indx;
float **a,b[];
{
    int i,ii=0,ip,j;
    float sum;

    for (i=1;i<=n;i++) {
        ip=indx[i];
        sum=b[ip];
        b[ip]=b[i];
        if (ii)
            for (j=ii;j<=i-1;j++) sum -= a[i][j]*b[j];
        else if (sum) ii=i;
        b[i]=sum;
    }
    for (i=n;i>=1;i--) {
        sum=b[i];
        for (j=i+1;j<=n;j++) sum -= a[i][j]*b[j];
        b[i]=sum/a[i][i];
    }
}

/* #include <malloc.h> */
#include <stdio.h>

void nrerror(error_text)
char error_text[];
{
    /*void exit();*/
```

```
    fprintf(stderr,"Numerical Recipes run-time error...\n");
    fprintf(stderr,"%s\n",error_text);
    fprintf(stderr,"...now exiting to system...\n");
    exit(1);
}



float *vector(nl,nh)
int nl,nh;
{
    float *v;

    v=(float *)malloc((unsigned) (nh-nl+1)*sizeof(float));
    if (!v) nrerror("allocation failure in vector()");
    return v-nl;
}

int *ivector(nl,nh)
int nl,nh;
{
    int *v;

    v=(int *)malloc((unsigned) (nh-nl+1)*sizeof(int));
    if (!v) nrerror("allocation failure in ivector()");
    return v-nl;
}

double *dvector(nl,nh)
int nl,nh;
{
    double *v;

    v=(double *)malloc((unsigned) (nh-nl+1)*sizeof(double));
    if (!v) nrerror("allocation failure in dvector()");
    return v-nl;
}



float **matrix(nrl,nrh,ncl,nch)
int nrl,nrh,ncl,nch;
```

```
{
    int i;
    float **m;

    m=(float **) malloc((unsigned) (nrh-nrl+1)*sizeof(float*));
    if (!m) nrerror("allocation failure 1 in matrix()");
    m -= nrl;

    for(i=nrl;i<=nrh;i++) {
        m[i]=(float *) malloc((unsigned) (nch-ncl+1)*sizeof(float));
        if (!m[i]) nrerror("allocation failure 2 in matrix()");
        m[i] -= ncl;
    }
    return m;
}

double **dmatrix(nrl,nrh,ncl,nch)
int nrl,nrh,ncl,nch;
{
    int i;
    double **m;

    m=(double **) malloc((unsigned) (nrh-nrl+1)*sizeof(double*));
    if (!m) nrerror("allocation failure 1 in dmatrix()");
    m -= nrl;

    for(i=nrl;i<=nrh;i++) {
        m[i]=(double *) malloc((unsigned) (nch-ncl+1)*sizeof(double));
        if (!m[i]) nrerror("allocation failure 2 in dmatrix()");
        m[i] -= ncl;
    }
    return m;
}

int **imatrix(nrl,nrh,ncl,nch)
int nrl,nrh,ncl,nch;
{
    int i,**m;

    m=(int **)malloc((unsigned) (nrh-nrl+1)*sizeof(int*));
    if (!m) nrerror("allocation failure 1 in imatrix()");
    m -= nrl;
```

```
    for(i=nrl;i<=nrh;i++) {
        m[i]=(int *)malloc((unsigned) (nch-ncl+1)*sizeof(int));
        if (!m[i]) nrerror("allocation failure 2 in imatrix()");
        m[i] -= ncl;
    }
    return m;
}


float **submatrix(a,oldrl,oldrh,oldcl,oldch,newrl,newcl)
float **a;
int oldrl,oldrh,oldcl,oldch,newrl,newcl;
{
    int i,j;
    float **m;

    m=(float **) malloc((unsigned) (oldrh-oldrl+1)*sizeof(float*));
    if (!m) nrerror("allocation failure in submatrix()");
    m -= newrl;

    for(i=oldrl,j=newrl;i<=oldrh;i++,j++) m[j]=a[i]+oldcl-newcl;

    return m;
}


void free_vector(v,nl,nh)
float *v;
int nl,nh;
{
    free((char*) (v+nl));
}

void free_ivector(v,nl,nh)
int *v,nl,nh;
{
    free((char*) (v+nl));
}
```

```
void free_dvector(v,nl,nh)
double *v;
int nl,nh;
{
    free((char*) (v+nl));
}



void free_matrix(m,nrl,nrh,ncl,nch)
float **m;
int nrl,nrh,ncl,nch;
{
    int i;

    for(i=nrh;i>=nrl;i--) free((char*) (m[i]+ncl));
    free((char*) (m+nrl));
}

void free_dmatrix(m,nrl,nrh,ncl,nch)
double **m;
int nrl,nrh,ncl,nch;
{
    int i;

    for(i=nrh;i>=nrl;i--) free((char*) (m[i]+ncl));
    free((char*) (m+nrl));
}

void free_imatrix(m,nrl,nrh,ncl,nch)
int **m;
int nrl,nrh,ncl,nch;
{
    int i;

    for(i=nrh;i>=nrl;i--) free((char*) (m[i]+ncl));
    free((char*) (m+nrl));
}



void free_submatrix(b,nrl,nrh,ncl,nch)
```

```
float **b;
int nrl,nrh,ncl,nch;
{
    free((char*) (b+nrl));
}



float **convert_matrix(a,nrl,nrh,ncl,nch)
float *a;
int nrl,nrh,ncl,nch;
{
    int i,j,nrow,ncol;
    float **m;

    nrow=nrh-nrl+1;
    ncol=nch-ncl+1;
    m = (float **) malloc((unsigned) (nrow)*sizeof(float*));
    if (!m) nrerror("allocation failure in convert_matrix()");
    m -= nrl;
    for(i=0,j=nrl;i<=nrow-1;i++,j++) m[j]=a+ncol*i-ncl;
    return m;
}



void free_convert_matrix(b,nrl,nrh,ncl,nch)
float **b;
int nrl,nrh,ncl,nch;
{
    free((char*) (b+nrl));
}
```

Verification Program

```
/*
     This program reads in the output of krige.c and verifies the means
     and variances to ensure numerical problems weren't encountered.
     If the values are wrong, new estimates are obtained by kriging
     the neighboring values.

     To run this program, type:

     verify.x <residuals fn> <variance fn> <mean.out fn> <var.out fn>

     Date of last modification:  22 JAN 90
*/

#include <stdio.h>
#include<math.h>
#include "nr.h"
#include "nrutil.h"

/*
     The following are the define parameters.

     NROW,NCOL: number of increments on the x,y axes
     MAXMN: the tolerance on the kriged residuals
     DELTA: the parameter which defines the neighborhood for kriging
     XMAX,YMAX: maximum values for x,y
     XMIN,YMIN: minimum values for x,y
     DX,DY: the length of the increments for the x,y axes
     KX,KY: scale parameters for x,y axes
     MAXPTS: the maximum number of points in a zone
*/

#define NROW 100
#define NCOL 50
#define MAXMN 50
#define DELTA 5
#define XMAX 4.
#define XMIN 0.
#define YMAX 300.
#define YMIN 100.
#define DX (XMAX-XMIN)/NROW
#define DY (YMAX-YMIN)/NCOL
```

```
#define KX (1.0)/(DX)
#define KY (1.0)/(DY)
#define MAXPTS 200

/*
     The following are global variables.

     num: the number of sampled points
     Sample[][]: the known points in a zone
     MatA[][]: the A matrix in the AX=B format for the kriging equations
     MatB[]: the B matrix in the AX=B format for the kriging equations
     MatX[]: the X matrix in the AX=B format for the kriging equations
*/

int num;
float Sample[MAXPTS][3];
float MatA[MAXPTS+3][MAXPTS+3];
float MatB[MAXPTS+3];
float MatX[MAXPTS+3];

main(argc,argv)
     int argc;
     char *argv[];

/*
     This is the main portion of the program.

     The following are local variables.

     i,j,k,irow,icol,ii,jj: loop variables
     numbad: counts the number of points which are out of tolerance
     iilow,iihigh,jjlow,jjhigh: ranges for loop variables ii,jj
     x,y: grid point coordinates
     sum,varsum: temporary variables
     Grid[][][]: output means and variances
     Mean[][]: input means
     Var[][]: input variances
*/

{
     int i,j,k,irow,icol,numbad;
     int ii,iilow,iihigh,jj,jjlow,jjhigh;
```

```
float x,y;
float sum,varsum;
float Grid[2][NROW][NCOL];
float Mean[NROW][NCOL];
float Var[NROW][NCOL];

void Build_A();
void Build_B();
void Build_X();

FILE *f_mnin, *f_varin, *f_mnout, *f_varout;

f_mnin = fopen(argv[1],"r");
f_varin = fopen(argv[2],"r");
f_mnout = fopen(argv[3],"w");
f_varout = fopen(argv[4],"w");

for(i=0; i<NROW; i++) {
   for(j=0; j<NCOL; j++) {
      Grid[0][i][j] = 0.0;
      Grid[1][i][j] = 0.0;
      Mean[i][j] = 0.0;
      Var[i][j] = 0.0;
   }
}

for(i=0; i<NROW; i++) {
   for(j=0; j<NCOL/10; j++) {

   fscanf(f_mnin,"%f %f %f %f %f %f %f %f %f %f\n",&Mean[i][j*10],
   &Mean[i][j*10+1],&Mean[i][j*10+2],&Mean[i][j*10+3],&Mean[i][j*10+4],
   &Mean[i][j*10+5],&Mean[i][j*10+6],&Mean[i][j*10+7],&Mean[i][j*10+8],
   &Mean[i][j*10+9]);

   }
}

for(i=0; i<NROW; i++) {
   for(j=0; j<NCOL/10; j++) {

   fscanf(f_varin,"%f %f %f %f %f %f %f %f %f %f\n",&Var[i][j*10],
   &Var[i][j*10+1],&Var[i][j*10+2],&Var[i][j*10+3],&Var[i][j*10+4],
```

```
    &Var[i][j*10+5],&Var[i][j*10+6],&Var[i][j*10+7],&Var[i][j*10+8],
    &Var[i][j*10+9]);

    }
}

for(i=0; i<NROW; i++) {
    for(j=0; j<NCOL; j++) {
        if((Mean[i][j]<MAXMN)&&(Mean[i][j]>-MAXMN)) {
            Grid[0][i][j]=Mean[i][j];
            Grid[1][i][j]=Var[i][j];
        }
        else {

            numbad++;
            printf("Correcting point %d , %d\n",i,j);

            iilow = i - DELTA;
            jjlow = j - DELTA;
            iihigh = i + DELTA;
            jjhigh = j + DELTA;

            if(i-DELTA<0) iilow = 0;
            if(j-DELTA<0) jjlow = 0;
            if(i+DELTA>NROW) iihigh = NROW;
            if(j+DELTA>NCOL) jjhigh = NCOL;

            num = 0;

            for(ii=iilow; ii<=iihigh; ii++) {
              for(jj=jjlow; jj<=jjhigh; jj++) {
                if((Mean[ii][jj]!=0)&&(Mean[ii][jj]<MAXMN)&&(Mean[ii][jj]>
                    -MAXMN)) {
                    Sample[num][0] = (ii+.5)*DX+XMIN;
                    Sample[num][1] = (jj+.5)*DY+YMIN;
                    Sample[num][2] = Mean[ii][jj];
                    num++;
                }
              }
            }

            Build_A();
```

```
                Build_B(i,j);
                Build_X();

                sum = 0.0;
                varsum = 0.0;

                for(ii=0; ii<num; ii++) {
                   sum += MatX[ii] * Sample[ii][2];
                   varsum += MatX[ii] * MatB[ii];
                }

                Grid[0][i][j] = sum;
                Grid[1][i][j] = varsum;

             }
          }
       }

       for(i=0; i<NROW; i++) {
          for(j=0; j<=NCOL-10; j=j+10) {
             for(k=0; k<10; k++) {
                fprintf(f_mnout,"%f ",Grid[0][i][j+k]);
             }
             fprintf(f_mnout,"\n");
          }
       }

       for(i=0; i<NROW; i++) {
          for(j=0; j<=NCOL-10; j=j+10) {
             for(k=0; k<10; k++) {
                fprintf(f_varout,"%f ",Grid[1][i][j+k]);
             }
             fprintf(f_varout,"\n");
          }
       }
       printf("Number of points corrected = %d \n",numbad);

       fclose(f_mnin);
       fclose(f_mnout);
       fclose(f_varin);
       fclose(f_varout);
    }
```

```
void Build_A()

/*
     This routine builds the A matrix of the kriging equations.

     The following are local variables.

     i,j: loop variables
     temp1,temp2: temporary variables
     h: the distance between any two points
*/

{
     int i,j;
     float temp1,temp2,h;

     /*
       This portion completes the gamma structure of A.
     */


     for(i=0; i<num; i++) {
          MatA[i][i]=0.0;
          for(j=0; j<num; j++) {
            temp1 = (Sample[i][0]-Sample[j][0]);
            temp2 = (Sample[i][1]-Sample[j][1]);
            h = sqrt(KX*KX*temp1*temp1+KY*KY*temp2*temp2);
            MatA[i][j]= h;
            MatA[j][i]=MatA[i][j];
          }
     }

     /*
       This portion adds a column and row of 1's.
     */

     for(i=0; i<num; i++) {
       MatA[i][num] = 1.0;
       MatA[num][i] = 1.0;
     }
```

```
    /*
      This portion provides terms for a trend.
    */

    for(i=0; i<num; i++) {
        for(j=0; j<2; j++) {
           MatA[i][num+1+j]=Sample[i][j];
           MatA[num+j+1][i]=Sample[i][j];
        }
    }

    /*
      This portion completes A with a block of 0's.
    */

    for(i=0; i<3; i++) {
      for(j=0; j<3; j++) {
           MatA[num+i][num+j] = 0.0;
      }
    }
}

void Build_B(i,j)
int i,j;

/*
    This routine builds the B matrix of the kriging equations.

    The following are local variables.

    ii: loop variable
    x: x coordinate of the (i,j) block midpoint
    y: y coordinate of the (i,j) block midpoint
    h: the distance between any two points
    temp1,temp2: temporary variables
*/

{
    int ii;
    float x,y,h,temp1,temp2;

    x = ((i+.5)*DX) + XMIN;
```

```
    y = ((j+.5)*DY) + YMIN;

    for(ii=0; ii<num; ii++) {
      temp1 = (x-Sample[ii][0]);
      temp2 = (y-Sample[ii][1]);
      h = sqrt(KX*KX*temp1*temp1+KY*KY*temp2*temp2);
      MatB[ii] = h;
    }

    MatB[num] = 1.0;
    MatB[num+1] = x;
    MatB[num+2] = y;

}




void Build_X()

/*
    This routine builds the X matrix of the kriging equations.

    The following variables are local variables.

    i,j: loop variables
    *indx, p, **a, *z, **c: "lu" variables
*/

{
    int i,j;
    int *indx;
    float p, **a, *z, **c;

    indx = ivector(1,num+3);
    a=matrix(1,num+3,1,num+3);
    z=vector(1,num+3);
    c=matrix(1,num+3,1,num+3);

    for(i=0; i<num+3; i++) {
      for(j=0; j<num+3; j++) {
```

```
            a[i+1][j+1] = MatA[i][j];
        }
    }

    ludcmp(a,num+3,indx,&p);

    for(i=0; i<num+3; i++) {
        z[i+1] = MatB[i];
    }

    lubksb(a,num+3,indx,z);

    for(i=0; i<num+3; i++) {
        MatX[i] = z[i+1];
    }
    free_matrix(a,1,num+3,1,num+3);
    free_matrix(c,1,num+3,1,num+3);
}


/*
    The following routines were adapted from Numerical Recipes
    for C.
*/

/*************************************************************/
#define TINY 1.0e-20;

void ludcmp(a,n,indx,d)
float **a;
int n,*indx;
float *d;
{
    int i,imax,j,k;
    float big,dum,sum,temp;
    float *vv,*vector();
    void nrerror(),free_vector();

    vv=vector(1,n);

    *d=1.0;
```

```
    for (i=1;i<=n;i++) {
        big=0.0;
        for (j=1;j<=n;j++)
            {
            if ((temp=fabs(a[i][j])) > big) big=temp;
            }
/*          if (big == 0.0) nrerror("Singular matrix in routine LUDCMP"); */
        if (big == 0.0) printf("Singular matrix in routine LUDCMP\n");
        vv[i]=1.0/big;
    }
    for (j=1;j<=n;j++) {
        for (i=1;i<j;i++) {
            sum=a[i][j];
            for (k=1;k<i;k++) sum -= a[i][k]*a[k][j];
            a[i][j]=sum;
        }
        big=0.0;
        for (i=j;i<=n;i++) {
            sum=a[i][j];
            for (k=1;k<j;k++)
                sum -= a[i][k]*a[k][j];
            a[i][j]=sum;
            if ( (dum=vv[i]*fabs(sum)) >= big) {
                big=dum;
                imax=i;
            }
        }
        if (j != imax) {
            for (k=1;k<=n;k++) {
                dum=a[imax][k];
                a[imax][k]=a[j][k];
                a[j][k]=dum;
            }
            *d = -(*d);
            vv[imax]=vv[j];
        }
        indx[j]=imax;
        if (a[j][j] == 0.0) a[j][j]=TINY;
        if (j != n) {
            dum=1.0/(a[j][j]);
            for (i=j+1;i<=n;i++) a[i][j] *= dum;
        }
```

```
    }
    free_vector(vv,1,n);
}

#undef TINY

void lubksb(a,n,indx,b)
int n,*indx;
float **a,b[];
{
    int i,ii=0,ip,j;
    float sum;

    for (i=1;i<=n;i++) {
        ip=indx[i];
        sum=b[ip];
        b[ip]=b[i];
        if (ii)
            for (j=ii;j<=i-1;j++) sum -= a[i][j]*b[j];
        else if (sum) ii=i;
        b[i]=sum;
    }
    for (i=n;i>=1;i--) {
        sum=b[i];
        for (j=i+1;j<=n;j++) sum -= a[i][j]*b[j];
        b[i]=sum/a[i][i];
    }
}

/* #include <malloc.h> */
#include <stdio.h>

void nrerror(error_text)
char error_text[];
{
    void exit();

    fprintf(stderr,"Numerical Recipes run-time error...\n");
    fprintf(stderr,"%s\n",error_text);
    fprintf(stderr,"...now exiting to system...\n");
    exit(1);
}
```

```
float *vector(nl,nh)
int nl,nh;
{
     float *v;

     v=(float *)malloc((unsigned) (nh-nl+1)*sizeof(float));
     if (!v) nrerror("allocation failure in vector()");
     return v-nl;
}

int *ivector(nl,nh)
int nl,nh;
{
     int *v;

     v=(int *)malloc((unsigned) (nh-nl+1)*sizeof(int));
     if (!v) nrerror("allocation failure in ivector()");
     return v-nl;
}

double *dvector(nl,nh)
int nl,nh;
{
     double *v;

     v=(double *)malloc((unsigned) (nh-nl+1)*sizeof(double));
     if (!v) nrerror("allocation failure in dvector()");
     return v-nl;
}


float **matrix(nrl,nrh,ncl,nch)
int nrl,nrh,ncl,nch;
{
     int i;
     float **m;

     m=(float **) malloc((unsigned) (nrh-nrl+1)*sizeof(float*));
```

```
    if (!m) nrerror("allocation failure 1 in matrix()");
    m -= nrl;

    for(i=nrl;i<=nrh;i++) {
        m[i]=(float *) malloc((unsigned) (nch-ncl+1)*sizeof(float));
        if (!m[i]) nrerror("allocation failure 2 in matrix()");
        m[i] -= ncl;
    }
    return m;
}

double **dmatrix(nrl,nrh,ncl,nch)
int nrl,nrh,ncl,nch;
{
    int i;
    double **m;

    m=(double **) malloc((unsigned) (nrh-nrl+1)*sizeof(double*));
    if (!m) nrerror("allocation failure 1 in dmatrix()");
    m -= nrl;

    for(i=nrl;i<=nrh;i++) {
        m[i]=(double *) malloc((unsigned) (nch-ncl+1)*sizeof(double));
        if (!m[i]) nrerror("allocation failure 2 in dmatrix()");
        m[i] -= ncl;
    }
    return m;
}

int **imatrix(nrl,nrh,ncl,nch)
int nrl,nrh,ncl,nch;
{
    int i,**m;

    m=(int **)malloc((unsigned) (nrh-nrl+1)*sizeof(int*));
    if (!m) nrerror("allocation failure 1 in imatrix()");
    m -= nrl;

    for(i=nrl;i<=nrh;i++) {
        m[i]=(int *)malloc((unsigned) (nch-ncl+1)*sizeof(int));
        if (!m[i]) nrerror("allocation failure 2 in imatrix()");
        m[i] -= ncl;
```

```
    }
    return m;
}


float **submatrix(a,oldrl,oldrh,oldcl,oldch,newrl,newcl)
float **a;
int oldrl,oldrh,oldcl,oldch,newrl,newcl;
{
    int i,j;
    float **m;

    m=(float **) malloc((unsigned) (oldrh-oldrl+1)*sizeof(float*));
    if (!m) nrerror("allocation failure in submatrix()");
    m -= newrl;

    for(i=oldrl,j=newrl;i<=oldrh;i++,j++) m[j]=a[i]+oldcl-newcl;

    return m;
}


void free_vector(v,nl,nh)
float *v;
int nl,nh;
{
    free((char*) (v+nl));
}

void free_ivector(v,nl,nh)
int *v,nl,nh;
{
    free((char*) (v+nl));
}

void free_dvector(v,nl,nh)
double *v;
int nl,nh;
{
    free((char*) (v+nl));
```

```
}


void free_matrix(m,nrl,nrh,ncl,nch)
float **m;
int nrl,nrh,ncl,nch;
{
     int i;

     for(i=nrh;i>=nrl;i--) free((char*) (m[i]+ncl));
     free((char*) (m+nrl));
}

void free_dmatrix(m,nrl,nrh,ncl,nch)
double **m;
int nrl,nrh,ncl,nch;
{
     int i;

     for(i=nrh;i>=nrl;i--) free((char*) (m[i]+ncl));
     free((char*) (m+nrl));
}

void free_imatrix(m,nrl,nrh,ncl,nch)
int **m;
int nrl,nrh,ncl,nch;
{
     int i;

     for(i=nrh;i>=nrl;i--) free((char*) (m[i]+ncl));
     free((char*) (m+nrl));
}


void free_submatrix(b,nrl,nrh,ncl,nch)
float **b;
int nrl,nrh,ncl,nch;
{
     free((char*) (b+nrl));
}
```

```
float **convert_matrix(a,nrl,nrh,ncl,nch)
float *a;
int nrl,nrh,ncl,nch;
{
     int i,j,nrow,ncol;
     float **m;

     nrow=nrh-nrl+1;
     ncol=nch-ncl+1;
     m = (float **) malloc((unsigned) (nrow)*sizeof(float*));
     if (!m) nrerror("allocation failure in convert_matrix()");
     m -= nrl;
     for(i=0,j=nrl;i<=nrow-1;i++.j++) m[j]=a+ncol*i-ncl;
     return m;
}


void free_convert_matrix(b,nrl,nrh,ncl,nch)
float **b;
int nrl,nrh,ncl,nch;
{
     free((char*) (b+nrl));
}
```

Program for Inclusion of Trend

```
/*
     This program reads in the output of krige.c and the mean face and
     combines them to obtain the surface estimates.

     To run this program, type:

      add_trend.x <subject fn> <mean face fn> <output fn>

*/

#include <stdio.h>
#include<math.h>

#define NROW 100
#define NCOL 50

main(argc,argv)
     int argc;
     char *argv[];

{
     int i,j,k,irow,icol,NUM;
     float x,y,z,d1,d2;
     float Grid[NROW][NCOL];
     float Mean_Face[NROW][NCOL];
     float Subject[NROW][NCOL];

     FILE *f_face, *f_mean, *fout;

     f_face = fopen(argv[1],"r");
     f_mean = fopen(argv[2],"r");
     fout = fopen(argv[3],"w");

     NUM = NROW * NCOL;

     for(i=0; i<NROW; i++) {
       for(j=0; j<NCOL; j++) {
          Grid[i][j] = 0.0;
          Mean_Face[i][j] = 0.0;
          Subject[i][j] = 0.0;
       }
```

```
    }

    for(i=0; i<NROW; i++) {
       for(j=0; j<NCOL/10; j++) {

      fscanf(f_face,"%f %f %f %f %f %f %f %f %f %f\n",&Subject[i][j*10],
      &Subject[i][j*10+1],&Subject[i][j*10+2],&Subject[i][j*10+3],
      &Subject[i][j*10+4],&Subject[i][j*10+5],&Subject[i][j*10+6],
      &Subject[i][j*10+7],&Subject[i][j*10+8],&Subject[i][j*10+9]);

       }
    }

    for(i=0; i<NROW; i++) {
       for(j=0; j<NCOL/10; j++) {

      fscanf(f_mean,"%f %f %f %f %f %f %f %f %f %f\n",&Mean_Face[i][j*10],
      &Mean_Face[i][j*10+1],&Mean_Face[i][j*10+2],&Mean_Face[i][j*10+3],
      &Mean_Face[i][j*10+4],&Mean_Face[i][j*10+5],&Mean_Face[i][j*10+6],
      &Mean_Face[i][j*10+7],&Mean_Face[i][j*10+8],&Mean_Face[i][j*10+9]);

       }
    }

    for(i=0; i<NROW; i++) {
       for(j=0; j<NCOL; j++) {
         Grid[i][j]=Mean_Face[i][j]+Subject[i][j];
       }
    }

    for(i=0; i<NROW; i++) {
      for(j=0; j<=NCOL-10; j=j+10) {
         for(k=0; k<10; k++) {
            fprintf(fout,"%f ",Grid[i][j+k]);
         }
         fprintf(fout,"\n");
       }
    }
    fclose(f_face);
    fclose(f_mean);
    fclose(fout);
}
```

Appendix I. *Bayesian Analysis Programs*

This appendix includes the program for updating the means and variances obtained through the kriging process. The execution command is provided in the initial comments. The second program creates the data file of initial variances.

C Updating Program

```
/*
     This program reads in the output of krige.c and the current
     estimate of the surface and combines them into the new surface
     estimate.

     To run this program, type:

      update.x <surfm> <surfv> <subm> <subv> <outm> <outv>

*/

#include <stdio.h>
#include<math.h>

#define NROW 100
#define NCOL 50

main(argc,argv)
     int argc;
     char *argv[];

{
     int i,j,k,irow,icol,NUM;
     float x,y,z,d1,d2,K;
     float Grid[2][NROW][NCOL];
     float Surface[2][NROW][NCOL];
     float Subject[2][NROW][NCOL];

     FILE *f_surfm, *f_surfv, *f_subm, *f_subv, *f_outm, *f_outv;

     f_surfm = fopen(argv[1],"r");
     f_surfv = fopen(argv[2],"r");
     f_subm = fopen(argv[3],"r");
     f_subv = fopen(argv[4],"r");
     f_outm = fopen(argv[5],"w");
     f_outv = fopen(argv[6],"w");

     for(i=0; i<2; i++) {
        for(j=0; j<NROW; j++) {
           for(k=0; k<NCOL; k++) {
              Surface[i][j][k] = 0.0;
```

```
          Subject[i][j][k] = 0.0;
          Grid[i][j][k] = 0.0;
        }
      }
    }

    for(i=0; i<NROW; i++) {
      for(j=0; j<NCOL/10; j++) {
      fscanf(f_surfm,"%f %f %f %f %f %f %f %f %f %f\n",&Surface[0][i][j*10],
     &Surface[0][i][j*10+1],&Surface[0][i][j*10+2],&Surface[0][i][j*10+3],
     &Surface[0][i][j*10+4],&Surface[0][i][j*10+5],&Surface[0][i][j*10+6],
     &Surface[0][i][j*10+7],&Surface[0][i][j*10+8],&Surface[0][i][j*10+9]);
      }
    }

    for(i=0; i<NROW; i++) {
      for(j=0; j<NCOL/10; j++) {
      fscanf(f_surfv,"%f %f %f %f %f %f %f %f %f %f\n",&Surface[1][i][j*10],
     &Surface[1][i][j*10+1],&Surface[1][i][j*10+2],&Surface[1][i][j*10+3],
     &Surface[1][i][j*10+4],&Surface[1][i][j*10+5],&Surface[1][i][j*10+6],
     &Surface[1][i][j*10+7],&Surface[1][i][j*10+8],&Surface[1][i][j*10+9]);
      }
    }

    for(i=0; i<NROW; i++) {
      for(j=0; j<NCOL/10; j++) {
      fscanf(f_subm,"%f %f %f %f %f %f %f %f %f %f\n",&Subject[0][i][j*10],
     &Subject[0][i][j*10+1],&Subject[0][i][j*10+2],&Subject[0][i][j*10+3],
     &Subject[0][i][j*10+4],&Subject[0][i][j*10+5],&Subject[0][i][j*10+6],
     &Subject[0][i][j*10+7],&Subject[0][i][j*10+8],&Subject[0][i][j*10+9]);
      }
    }

    for(i=0; i<NROW; i++) {
      for(j=0; j<NCOL/10; j++) {
      fscanf(f_subv,"%f %f %f %f %f %f %f %f %f %f\n",&Subject[1][i][j*10],
     &Subject[1][i][j*10+1],&Subject[1][i][j*10+2],&Subject[1][i][j*10+3],
     &Subject[1][i][j*10+4],&Subject[1][i][j*10+5],&Subject[1][i][j*10+6],
     &Subject[1][i][j*10+7],&Subject[1][i][j*10+8],&Subject[1][i][j*10+9]);
      }
    }
```

```
    for(i=0; i<NROW; i++) {
     for(j=0; j<NCOL; j++) {
      if(Subject[1][i][j]>0.0) {
           K=Surface[1][i][j]/(Surface[1][i][j]+Subject[1][i][j]);
           Grid[0][i][j]=Surface[0][i][j]+K*(Subject[0][i][j]-Surface[0][i][j]);
           Grid[1][i][j]=Surface[1][i][j]-K*Surface[1][i][j];
      }
      else {
           Grid[0][i][j]=Surface[0][i][j];
           Grid[1][i][j]=Surface[1][i][j];
      }
     }
    }

    for(i=0; i<NROW; i++) {
       for(j=0; j<=NCOL-10; j=j+10) {
          for(k=0; k<10; k++) {
             fprintf(f_outm,"%f ",Grid[0][i][j+k]);
          }
          fprintf(f_outm,"\n");
       }
    }

    for(i=0; i<NROW; i++) {
       for(j=0; j<=NCOL-10; j=j+10) {
          for(k=0; k<10; k++) {
             fprintf(f_outv,"%f ",Grid[1][i][j+k]);

           printf(f_outv,"\n");
       }
    }

    fclose(f_surfm);
    fclose(f_surfv);
    fclose(f_subm);
    fclose(f_subv);
    fclose(f_outm);
    fclose(f_outv):
}
```

Program for Generating Initial Variances

```
/*
     This program creates and array of constant variances and writes
     them to a file called initial.var.

     To run the program, type:

     same_var.x

*/

#include <stdio.h>
#include<math.h>

#define NROW 100
#define NCOL 50
#define VAR 20.0

main(argc,argv)
     int argc;
     char *argv[];

{
     int i,j,k,irow.icol,NUM;
     float x,y,z,d1,d2;
     float Grid[NROW][NCOL];

     FILE *fout;

     fout = fopen("initial.var","w");

     NUM = NROW * NCOL;

     for(i=0; i<NROW; i++) {
        for(j=0; j<NCOL; j++) {
           Grid[i][j] = VAR;
        }
     }

     for(i=0; i<NROW; i++) {
        for(j=0; j<=NCOL-10; j=j+10) {
           for(k=0; k<10; k++) {
```

```
            fprintf(fout,"%f ",Grid[i][j+k]);
          }
          fprintf(fout,"\n");
        }
      }
      fclose(fout);
}
```

Appendix J. *Graphics Programs*

This appendix includes the Interactive Data Language (IDL) procedures which were used to plot the variograms and surfaces. IDL is a product of Research Systems, Inc.. More information on IDL is available in the *Introduction To IDL* and the *IDL User's Guide*. The following programs are only a sample of the routines which were used. However, these procedures sufficiently demonstrate the method for producing the plots.

Variogram Plotting Procedures

This section provides examples of the variogram plotting programs. The first program generates a postscript file for the variograms of Subject 09. The code is provided below.

```
set_plot,'ps'
device,font_size=20
device,/encapsulated,filename='figv09.ps'
device,/inches,xsize=3.5,scale_factor=0.9
device,/inches,ysize=3.5,scale_factor=0.9
N=10
M=10
XR=8
YR=10
a=fltarr(3,4*N)
f=fltarr(2,4,N)
g=fltarr(2,4,M)
openr,1,'v09.out'
readf,1,a
close,1
FOR K=0,1 DO BEGIN
FOR I=0,N-1 DO f(K,0,I)=a(K,I)
FOR I=0,N-1 DO f(K,1,I)=a(K,I+N)
FOR I=0,N-1 DO f(K,2,I)=a(K,I+2*N)
FOR I=0,N-1 DO f(K,3,I)=a(K,I+3*N)
FOR I=0,M-1 DO FOR J=0,3 DO g(K,J,I)=f(K,J,I)
ENDFOR
plot,g(0,0,*),g(1,0,*),xtitle='!8h (distance)',ytitle='!7c!8(h)',
xrange=[0,XR],yrange=[0,YR],line=1,xcharsize=1.3,ycharsize=1.3
oplot,g(0,1,*),g(1,1,*),line=2
oplot,g(0,2,*),g(1,2,*),line=3
oplot,g(0,3,*),g(1,3,*),line=4
sphere=fltarr(2,M+2)
FOR I=0,M+1 DO BEGIN
sphere(0,I)=I
IF (I GT 6.645) THEN BEGIN
sphere(1,I)=2.226+0.689
ENDIF ELSE BEGIN
sphere(1,I)=2.226*(1.5*I/6.645 - .5*I*I*I/293.394) + 0.689
ENDELSE;
ENDFOR
```

```
oplot,sphere(0,*),sphere(1,*),line=0
b=fltarr(2,2)
c=fltarr(2,2)
d=fltarr(2,2)
e=fltarr(2,2)
f=fltarr(2,2)
b(0,0)=3.00
b(0,1)=3.50
b(1,0)=9.3
b(1,1)=9.3
c(0,0)=3.00
c(0,1)=3.50
c(1,0)=9.1
c(1,1)=9.1
d(0,0)=3.00
d(0,1)=3.50
d(1,0)=9.2
d(1,1)=9.2
e(0,0)=3.00
e(0,1)=3.50
e(1,0)=9.00
e(1,1)=9.00
f(0,0)=3.00
f(0,1)=3.50
f(1,0)=8.40
f(1,1)=8.40
oplot,b(0,*),b(1,*),line=1
oplot,c(0,*),c(1,*),line=2
oplot,d(0,*),d(1,*),line=3
oplot,e(0,*),e(1,*),line=4
oplot,f(0,*),f(1,*),line=0
xyouts,3.8,9.0,'Subject 09',size=1.0
xyouts,3.8,8.2,'Estimated',size=1.0
device,/close_file
set_plot,'sun'
end
```

This next example is a program for plotting the variograms of a subject who does not follow the standard pattern. This sample is for Subject 01.

```
set_plot,'ps'
```

```
device,font_size=20
device,/encapsulated,filename='figv01.ps'
device,/inches,xsize=3.5,scale_factor=0.9
device,/inches,ysize=3.5,scale_factor=0.9
N=10
M=10
XR=8
YR=20
a=fltarr(3,4*N)
f=fltarr(2,4,N)
g=fltarr(2,4,M)
openr,1,'v01.out'
readf,1,a
close,1
FOR K=0,1 DO BEGIN
FOR I=0,N-1 DO f(K,0,I)=a(K,I)
FOR I=0,N-1 DO f(K,1,I)=a(K,I+N)
FOR I=0,N-1 DO f(K,2,I)=a(K,I+2*N)
FOR I=0,N-1 DO f(K,3,I)=a(K,I+3*N)
FOR I=0,M-1 DO FOR J=0,3 DO g(K,J,I)=f(K,J,I)
ENDFOR
plot,g(0,0,*),g(1,0,*),xtitle='!8h (distance)',ytitle='!7c!8(h)',
xrange=[0,XR],yrange=[0,YR],line=1,xcharsize=1.3,ycharsize=1.3
oplot,g(0,1,*),g(1,1,*),line=2
oplot,g(0,2,*),g(1,2,*),line=3
oplot,g(0,3,*),g(1,3,*),line=4
sphere=fltarr(2,M+2)
FOR I=0,M+1 DO BEGIN
sphere(0,I)=I
IF (I GT 6.645) THEN BEGIN
sphere(1,I)=2.226+0.689
ENDIF ELSE BEGIN
sphere(1,I)=2.226*(1.5*I/6.645 - .5*I*I*I/293.394) + 0.689
ENDELSE;
ENDFOR
oplot,sphere(0,*),sphere(1,*),line=0
b=fltarr(2,2)
c=fltarr(2,2)
d=fltarr(2,2)
e=fltarr(2,2)
f=fltarr(2,2)
b(0,0)=3.00
```

```
b(0,1)=3.50
b(1,0)=18.6
b(1,1)=18.6
c(0,0)=3.00
c(0,1)=3.50
c(1,0)=18.2
c(1,1)=18.2
d(0,0)=3.00
d(0,1)=3.50
d(1,0)=18.4
d(1,1)=18.4
e(0,0)=3.00
e(0,1)=3.50
e(1,0)=18.00
e(1,1)=18.00
f(0,0)=3.00
f(0,1)=3.50
f(1,0)=16.80
f(1,1)=16.80
oplot,b(0,*),b(1,*),line=1
oplot,c(0,*),c(1,*),line=2
oplot,d(0,*),d(1,*),line=3
oplot,e(0,*),e(1,*),line=4
oplot,f(0,*),f(1,*),line=0
xyouts,3.8,18.0,'Subject 01',size=1.0
xyouts,3.8,16.4,'Estimated',size=1.0
end
device,/close_file
set_plot,'sun'
end
```

The following program produces a postscript file for the 25 variograms which followed the same pattern.

```
S=''
set_plot,'ps'
device,font_size=20
device,/encapsulated,filename='fig25v.ps'
device,/inches,xsize=3.5,scale_factor=0.9
device,/inches,ysize=3.5,scale_factor=0.9
```

```
N=10
M=10
XR=8
YR=20
a=fltarr(3,4*N)
f=fltarr(2,4,N)
g=fltarr(2,4,M)
bb=fltarr(2,2)
cc=fltarr(2,2)
dd=fltarr(2,2)
ee=fltarr(2,2)
bb(0,0)=3.00
bb(0,1)=3.50
bb(1,0)=2*9.3
bb(1,1)=2*9.3
cc(0,0)=3.00
cc(0,1)=3.50
cc(1,0)=2*9.1
cc(1,1)=2*9.1
dd(0,0)=3.00
dd(0,1)=3.50
dd(1,0)=2*9.2
dd(1,1)=2*9.2
ee(0,0)=3.00
ee(0,1)=3.50
ee(1,0)=2*9.00
ee(1,1)=2*9.00
openr,2,'list.25'
readf,2,NUM
readf,2,S
openr,1,S
readf,1,a
close,1
FOR K=0,1 DO BEGIN
FOR I=0,N-1 DO f(K,0,I)=a(K,I)
FOR I=0,N-1 DO f(K,1,I)=a(K,I+N)
FOR I=0,N-1 DO f(K,2,I)=a(K,I+2*N)
FOR I=0,N-1 DO f(K,3,I)=a(K,I+3*N)
FOR I=0,M-1 DO FOR J=0,3 DO g(K,J,I)=f(K,J,I)
ENDFOR
plot,g(0,0,*),g(1,0,*),xtitle='!8h (distance)',ytitle='!7c!8(h)',
xrange=[0,XR],yrange=[0,YR],line=1,xcharsize=1.3,ycharsize=1.3
```

```
oplot,g(0,1,*),g(1,1,*),line=2
oplot,g(0,2,*),g(1,2,*),line=3
oplot,g(0,3,*),g(1,3,*),line=4
FOR LL=2,NUM DO BEGIN
readf,2,S
openr,1,S
readf,1,a
close,1
FOR K=0,1 DO BEGIN
FOR I=0,N-1 DO f(K,0,I)=a(K,I)
FOR I=0,N-1 DO f(K,1,I)=a(K,I+N)
FOR I=0,N-1 DO f(K,2,I)=a(K,I+2*N)
FOR I=0,N-1 DO f(K,3,I)=a(K,I+3*N)
FOR I=0,M-1 DO FOR J=0,3 DO g(K,J,I)=f(K,J,I)
ENDFOR
oplot,g(0,0,*),g(1,0,*),line=1
oplot,g(0,1,*),g(1,1,*),line=2
oplot,g(0,2,*),g(1,2,*),line=3
oplot,g(0,3,*),g(1,3,*),line=4
ENDFOR
close,2
oplot,bb(0,*),bb(1,*),line=1
oplot,cc(0,*),cc(1,*),line=2
oplot,dd(0,*),dd(1,*),line=3
oplot,ee(0,*),ee(1,*),line=4
xyouts,3.8,18.0,'25 Subjects',size=1.0
device,/close_file
set_plot,'sun'
end
```

This program is similar to the previous program except that it plots the variograms for 30 subjects.

```
S=''
set_plot,'ps'
device,font_size=20
device,/encapsulated,filename='fig30v.ps'
device,/inches,xsize=3.5,scale_factor=0.9
device,/inches,ysize=3.5,scale_factor=0.9
N=10
```

```
M=10
XR=8
YR=20
a=fltarr(3,4*N)
f=fltarr(2,4,N)
g=fltarr(2,4,M)
bb=fltarr(2,2)
cc=fltarr(2,2)
dd=fltarr(2,2)
ee=fltarr(2,2)
bb(0,0)=3.00
bb(0,1)=3.50
bb(1,0)=2*9.3
bb(1,1)=2*9.3
cc(0,0)=3.00
cc(0,1)=3.50
cc(1,0)=2*9.1
cc(1,1)=2*9.1
dd(0,0)=3.00
dd(0,1)=3.50
dd(1,0)=2*9.2
dd(1,1)=2*9.2
ee(0,0)=3.00
ee(0,1)=3.50
ee(1,0)=2*9.00
ee(1,1)=2*9.00
openr,2,'list.30'
readf,2,NUM
readf,2,S
openr,1,S
readf,1,a
close,1
FOR K=0,1 DO BEGIN
FOR I=0,N-1 DO f(K,0,I)=a(K,I)
FOR I=0,N-1 DO f(K,1,I)=a(K,I+N)
FOR I=0,N-1 DO f(K,2,I)=a(K,I+2*N)
FOR I=0,N-1 DO f(K,3,I)=a(K,I+3*N)
FOR I=0,M-1 DO FOR J=0,3 DO g(K,J,I)=f(K,J,I)
ENDFOR
plot,g(0,0,*),g(1,0,*),xtitle='!8h (distance)',ytitle='!7c!8(h)',
xrange=[0,XR],yrange=[0,YR],line=1,xcharsize=1.3,ycharsize=1.3
oplot,g(0,1,*),g(1,1,*),line=2
```

```
oplot,g(0,2,*),g(1,2,*),line=3
oplot,g(0,3,*),g(1,3,*),line=4
FOR LL=2,NUM DO BEGIN
readf,2,S
openr,1,S
readf,1,a
close,1
FOR K=0,1 DO BEGIN
FOR I=0,N-1 DO f(K,0,I)=a(K,I)
FOR I=0,N-1 DO f(K,1,I)=a(K,I+N)
FOR I=0,N-1 DO f(K,2,I)=a(K,I+2*N)
FOR I=0,N-1 DO f(K,3,I)=a(K,I+3*N)
FOR I=0,M-1 DO FOR J=0,3 DO g(K,J,I)=f(K,J,I)
ENDFOR
oplot,g(0,0,*),g(1,0,*),line=1
oplot,g(0,1,*),g(1,1,*),line=2
oplot,g(0,2,*),g(1,2,*),line=3
oplot,g(0,3,*),g(1,3,*),line=4
ENDFOR
close,2
oplot,bb(0,*),bb(1,*),line=1
oplot,cc(0,*),cc(1,*),line=2
oplot,dd(0,*),dd(1,*),line=3
oplot,ee(0,*),ee(1,*),line=4
xyouts,3.8,18.0,'30 Subjects',size=1.0
device,/close_file
set_plot,'sun'
end
```

Surface Plotting Procedures

This section includes programs which require an interactive input to specify the appropriate data file to plot. To obtain a hard copy of the output, the PLOT and PRINT programs must be used in conjunction with these procedures. To obtain an encapsulated postscript file, the OPEN and CLOSE procedures must be used. These files are discussed later in this section.

The first procedure plots any facial data set.

```
FN=''
READ,'Enter filename > ',FN
a=fltarr(50,100)
b=fltarr(40,80)
openr,1,FN
readf,1,a
close,1
for i=0,39 do begin
for j=0,79 do b(i,j)=a(10+i,5+j)
endfor
surface,b,az=45,ax=30,xrange=[0,40],yrange=[0,70],zrange=[0,200],
xtitle='!8Altitude',ytitle='Angle',ztitle='Radius',charsize=2.5
end
```

This next procedure plots residual data files.

```
FN=''
READ,'Enter filename > ',FN
a=fltarr(50,100)
b=fltarr(40,80)
openr,1,FN
readf,1,a
close,1
for i=0,39 do begin
for j=0,79 do b(i,j)=a(10+i,5+j)
endfor
m=20
surface,b,az=45,ax=30,xrange=[0,40],yrange=[0,70],zrange=[-m,m],
```

```
xtitle='!8Altitude',ytitle='Angle',ztitle='Residuals',charsize=2.5
end
```

The following program plots the variograms for an individual subject.

```
FN=''
READ,'Enter filename > ',FN
a=fltarr(50,100)
b=fltarr(40,80)
openr,1,FN
readf,1,a
close,1
for i=0,39 do begin
for j=0,79 do begin
if a(10+i,5+j) NE 20.0 then b(i,j)=a(10+i,5+j) else b(i,j)=0.0
endfor
endfor
surface,b,az=45,ax=30,xrange=[0,40],yrange=[0,70],zrange=[0,4],
xtitle='!8Altitude',ytitle='Angle',ztitle='Variance',charsize=2.5
end
```

Miscellaneous Procedures

The following miscellaneous procedures may be used to obtain hard copies, to produce encapsulated postscript files, or to set the terminal type to emulate a Tektronics terminal.

The first program, OPEN.PRO, is run prior to a sequence of plotting commands. Following this sequence, the CLOSE.PRO procedure is executed to produce the psotscript file. The combination of OPEN and CLOSE is necessary to produce the file and to return IDL to the appropriate environment settings. The open procedure is as follows.

```
set_plot,'ps'
FN=''
read,'Enter the name of the picture file > ',FN
device,/encapsulated,filename=FN
device,/inches,xsize=5.0,scale_factor=1.0
device,/inches,ysize=5.0,scale_factor=1.0
end
```

The close procedure is as follows.

```
device,/close_file
set_plot,'sun'
end
```

The plot and print procedures work in the same fashion as the two proceeding programs and produce a plot from the laser printer. The code for the plot procedure is as follows.

```
device,/close
cmd='lpr idl.ps'
spawn,cmd
set_plot,'sun'
end
```

The following procedure is used with the plot procedure to obtain the hard copy.

```
set_plot,'ps'
device,/inches,xsize=5.0,scale_factor=1.0
device,/inches,ysize=5.0,scale_factor=1.0
end
```

This last procedure allows the user to emulate a Tektronics terminal.

```
set_plot,'tek'
device,/tek4100,colors=8
end
```

Appendix K. *Multivariate Analysis Programs*

This appendix includes the C programs for extracting the data and the SAS code for performing the multivariate analysis. The first program reads the coordinates of the landmarks for each subject and writes them to a new file for input to the second program. This second program calculates the angles and distances and generates the SAS input file. Finally, the SAS code is provided to replicate the multivariate analysis.

Landmark Extraction Program

```
/*  This programs reads the points from the outXX.rlnd files
    in the /home2/robinson/Lndmrk/ directory into a face.XX
    file in the oper685 directory.
*/

#include <stdio.h>
#include <math.h>

main(argc,argv)
     int argc;
     char *argv[];

{
     int i,lndmrk[33];
     float x[33],y[33],z[33];
     int id1,id2;
     float d1,d2,d3,d4,d5,d6;

     FILE *fin, *fout;

     fin = fopen(argv[1],"r");
     fout = fopen(argv[2],"w");

     fscanf(fin,"%f %f %f %f %f %f",&d1,&d2,&d3,&d4,&d5,&d6);

     for(i=0; i<33; i++) {

      fscanf(fin,"%d %d %d %f %f %f %f",&lndmrk[i],&id1,&id2,&d1,
      &x[i],&y[i],&z[i]);

     }

     fprintf(fout,"%d %f %f %f\n",lndmrk[0],x[0],y[0],z[0]);
     fprintf(fout,"%d %f %f %f\n",lndmrk[2],x[2],y[2],z[2]);
     fprintf(fout,"%d %f %f %f\n",lndmrk[6],x[6],y[6],z[6]);
     fprintf(fout,"%d %f %f %f\n",lndmrk[11],x[11],y[11],z[11]);
     fprintf(fout,"%d %f %f %f\n",lndmrk[13],x[13],y[13],z[13]);
     fprintf(fout,"%d %f %f %f\n",lndmrk[14],x[14],y[14],z[14]);
     fprintf(fout,"%d %f %f %f\n",lndmrk[15],x[15],y[15],z[15]);
     fprintf(fout,"%d %f %f %f\n",lndmrk[16],x[16],y[16],z[16]);
     fprintf(fout,"%d %f %f %f\n",lndmrk[26],x[26],y[26],z[26]);
```

```
    fprintf(fout,"%d %f %f %f\n",lndmrk[29],x[29],y[29],z[29]);
    fprintf(fout,"%d %f %f %f\n",lndmrk[31],x[31],y[31],z[31]);

    fclose(fin);
    fclose(fout);

}
```

Distance and Angle Program

```
/*  This programs reads the points from the face.XX files and
    builds a file for the SAS cluster routines.  (sas.dat)
*/

#include <stdio.h>
#include <math.h>

#define NPTS 11

float d[NPTS][3];

main(argc,argv)
     int argc;
     char *argv[];

{
     int i,j,nosub;
     int lndmrk[NPTS];
     float t[26];
     char subject_name[10];
     float distance(),angle();

     FILE *fin, *fsub, *fout;

     fin = fopen(argv[1],"r");
     fout = fopen("sas.dat","w");

     fscanf(fin,"%d\n",&nosub);

     for(i=0; i<nosub; i++) {
        fscanf(fin,"%s\n",subject_name);
        fsub = fopen(subject_name,"r");

        for(j=0; j<NPTS; j++) {
        fscanf(fsub,"%d %f %f %f\n",&lndmrk[j],&d[j][0],&d[j][1],&d[j][2]);
        /* printf("%d   %f %f %f\n",j,d[j][0],d[j][1],d[j][2]); */
        }

    fclose(fsub);

    t[0]=i;
```

```
    t[1]=distance(2,8);
    t[2]=distance(0,10);
    t[3]=distance(1,9);
    t[4]=distance(3,11);
    t[5]=distance(3,4);
    t[6]=distance(7,5);
    t[7]=distance(4,6);
    t[8]=distance(3,6);
    t[9]=distance(7,4);
    t[10]=distance(3,5);
    t[11]=angle(0,3,10);
    t[12]=angle(0,5,10);
    t[13]=angle(1,3,9);
    t[14]=angle(1,6,9);
    t[15]=angle(2,4,8);
    t[16]=angle(2,7,8);
    t[17]=angle(1,4,9);
    t[18]=angle(3,4,5);
    t[19]=angle(3,4,7);
    t[20]=angle(3,4,6);
    t[21]=angle(0,7,10);
    t[22]=angle(0,6,10);
    t[23]=angle(2,6,8);
    t[24]=angle(2,5,8);
    t[25]=angle(1,7,9);

    fprintf(fout,"%f %f %f %f %f %f %f %f %f %f %f %f %f\n",t[0],
    t[1],t[2],t[3],t[4],t[5],t[6],t[7],t[8],t[9],t[10],t[11],t[12]);

    fprintf(fout,"%f %f %f %f %f %f %f %f %f %f %f %f %f\n",t[13],t[14],
    t[15],t[16],t[17],t[18],t[19],t[20],t[21],t[22],t[23],t[24],t[25]);

     }
     fclose(fin);
     fclose(fout);
}
float distance(p1,p2)
     int p1,p2;
{
     float sum;
     int i;
```

```
    sum =0.0;

    for(i=0; i<3; i++) {
       sum += (d[p1][i]-d[p2][i])*(d[p1][i]-d[p2][i]);
    }
    return sqrt(sum);
}

float angle(p1,p2,p3)
    int p1,p2,p3;
{
    int i;
    float a,b,c;
    float sum;

    sum=0.0;

    for(i=0; i<3; i++) {
       sum += (d[p2][i]-d[p3][i])*(d[p2][i]-d[p3][i]);
    }
    a=sqrt(sum);

    sum=0.0;

    for(i=0; i<3; i++) {
       sum += (d[p1][i]-d[p3][i])*(d[p1][i]-d[p3][i]);
    }
    b=sqrt(sum);

    sum=0.0;

    for(i=0; i<3; i++) {
       sum += (d[p1][i]-d[p2][i])*(d[p1][i]-d[p2][i]);
    }
    c=sqrt(sum);

    return acos((a*a+c*c-b*b)/(2.0*a*c));
}
```

Multivariate SAS Code

```
options linesize=78;
data dat;
infile dat;
input s v1-v25;
proc print;
proc corr;
  var v1-v25;
proc factor;
  var v1-v25;
proc factor;
  var v2-v3 v5-v11 v13-v20 v23-v25;
proc factor outstat=saveall;
  var v3 v5-v10 v13-v20 v23-v25;
proc factor data=saveall n=5 rotate=varimax score outstat=saves;
proc score data=dat score=saves out=savesc;
proc print;
   var factor1-factor5;
proc plot;
   plot factor2*factor1;
   plot factor3*factor1;
   plot factor4*factor1;
   plot factor5*factor1;
   plot factor3*factor2;
   plot factor4*factor2;
   plot factor5*factor2;
   plot factor4*factor3;
   plot factor5*factor3;
   plot factor5*factor4;
proc cluster method=average pseudo;
   var factor1-factor5;
/*proc tree n=5;*/
```

END

FILMED

DATE: 5-90

DTIC